எளிய தமிழில் Agile/Scrum

மென்பொருள் திட்ட மேலாண்மை



இரா. அசோகன்

எளிய தமிழில் Agile/Scrum

மென்பொருள் திட்ட மேலாண்மை

ஆசிரியர் :

இரா. அசோகன்

ashokramach@gmail.com

மின்னூல் வெளியீடு :

http://www.kaniyam.com/

அட்டைப்படம், மின்னூலாக்கம் :

பிரசன்னா

udpmprasanna@gmail.com

பொருளடக்கம்

Agile/Scrum - மென்பொருள் திட்ட மேலாண்மை

இதை, இந்த நூல் எளிமையாக அறிமுகம் செய்கிறது.

தமிழில் கட்டற்ற மென்பொருட்கள் பற்றிய தகவல்களை "கணியம்" மின் மாத இதழ், 2012 முதல் வெளியிட்டு வருகிறது. இதில் **"மென்பொருள் உருவாக்கும் விந்தையுலகம்"** *என்ற தலைப்பில் வெளியான* "Agile/Scrum" *பற்றிய கட்டுரைகளை இணைத்து ஒரு முழு புத்தகமாக வெளியிடுவதில் பெரு மகிழ்ச்சி கொள்கிறோம்.*

உங்கள் கருத்துகளையும், பிழை திருத்தங்களையும் editor@kaniyam.com - க்கு மின்னஞ்சல் அனுப்பலாம்.

படித்து பயன் பெறவும், பிறருடன் பகிர்ந்து மகிழவும் வேண்டுகிறோம்.

கணியம் இதழை தொடர்ந்து வளர்க்கும் அனைத்து அன்பர்களுக்கும் எமது நன்றிகள்.

த.சீனிவாசன்
tshrinivasan@gmail.com

ஆசிரியர்

கணியம்
editor@kaniyam.com

1. மென்பொருள் திட்டங்கள் பாதிக்கு மேல் படுதோல்வி அடைகின்றன!

நீங்கள் கடந்த இரண்டு ஆண்டுகளில் அமெரிக்காவில் ஒபாமா கேர் மருத்துவக் காப்பீடு திட்டத்தை செயல்படுத்துவதில் வந்த பிரச்சினைகள் பற்றி செய்திகள் பார்த்திருக்கக்கூடும். இத்திட்டத்தின்படி அமெரிக்காவில் குறைந்த வருமானம் உள்ள குடும்பங்கள் HealthCare.gov என்ற இணையதளத்தில் பதிவு செய்து தனியார் மருத்துவக் காப்பீடுகளை ஒப்பிட்டு, தேர்வுசெய்து வாங்கவும் மற்றும் அதற்கான அரசாங்க மானியம் பெறவும் இயலும். இந்த திட்டத்துக்கு ஆன மொத்த செலவு ரூபாய் 10,000 கோடி.

பிரச்சினைகள் வந்தவுடன் பல நிபுணர்களைக் கலந்து ஆலோசித்து அதிரடி நடவடிக்கைகள் எடுத்து சரி செய்துவிட்டார்கள். கொஞ்சம் தாமதம், நிறைய செலவு – ஆனால் வேலை நடக்கிறது. குறைந்த வருமானம் உள்ள சுமார் ஒரு கோடி குடும்பங்களுக்கு மருத்துவக் காப்பீடு கிடைக்கிறது.

ஆனால் இங்கிலாந்தில் *National Health Service (NHS)* ஆரம்பித்த நோயாளி ஆவண அமைப்புக்கு *(patient record system)* அந்த பாக்கியம் கிட்டவில்லை. கார்டியன் செய்திப்படி மக்களுடைய வரிப்பணத்தில் ரூபாய் 100,000 கோடிக்குமேல் செலவு செய்தபின் வேலைக்கு ஆகாது என்று கைவிடப்பட்டது. ஒப்பிட்டுப் பார்க்கப்போனால் சென்னை மெட்ரோ செலவு மதிப்பீடு சுமார் 20,000 கோடி ரூபாய். ஒன்றல்ல, இரண்டல்ல, ஐந்து சென்னை மெட்ரோக்கள்

கட்டும் அளவு செலவு. கைவிடப்பட்டபின் மிச்சம் ஒன்றும் இல்லை. இதுதான் விருப்பிற்கேற்ற மென்பொருளின் தனிச்சிறப்பு. நீங்கள் அதை உடன் பயன்படுத்தவில்லை என்றால் ஜீபூம்பா மந்திரம் போட்டதுபோல செய்த செலவெல்லாம் மாயமாய் மறைந்துவிடும்!

இம்மாதிரி ஒரு மென்பொருள் திட்டத்தை பெருந்தொகை செலவு செய்தபின் தோல்வியடைந்து கைவிடுவது எப்போதோ நடக்கும் அரிய விதிவிலக்கா? இல்லை! ஆலோசனை நிறுவனம் Standish Group-இடம் 50,000 திட்டங்கள் பற்றிய ஒரு தரவுத்தளம் உள்ளது. இவற்றில் 2003 முதல் 2012 வரை ரூபாய் 60 கோடிக்கு மேல் செலவாகும் 3500 பெரிய மென்பொருள் திட்டங்களில் 42% திட்டங்கள் கைவிடப்பட்டன அல்லது திரும்பவும் முதலில் இருந்து புதிதாகத் தொடங்கப்பட்டன. மற்றும் 51% திட்டங்கள் கெடுவில் முடியவில்லை,

செலவு பட்ஜெட்டுக்கு மேல் அல்லது எதிர்பார்த்த அளவில் பயன் தரவில்லை.



இவை அரசாங்க பொதுத்துறைத் திட்டங்கள். மக்களின் வரிப்பணத்தில் லஞ்சம், ஊழல் போலிருக்கிறது என்று நீங்கள் நினைக்கலாம். தனியார் துறையும் இதே லட்சணம் தான்.

- ஹெர்ஷே நிறுவனம் வட அமெரிக்காவிலும் மற்ற பல நாடுகளிலும் மிகப் பெரிய சாக்லேட் உற்பத்தியாளர். 1996-ல் இந்த நிறுவனம் ஒரு ஆர்டர் எடுத்து நிறைவேற்றும் புதிய அமைப்புக்கு மாற்றம் செய்ய ஆரம்பித்தது. இதன் விளைவாக ஹாலோவீன் திருவிழா நேரத்தில் 600 கோடி ரூபாய் மதிப்புள்ள மிட்டாய்களை ஆர்டர் கொடுத்தவர்களுக்கு உறுதி செய்தபடி அனுப்ப இயலவில்லை.
- கே-மார்ட் அமெரிக்காவில் மிகவும் பிரபலமான சில்லறை விற்பனையாளர். இந்த நிறுவனம் 2000-ஆம் ஆண்டில், 8500 கோடி ரூபாய் முதலீட்டில் தகவல் தொழில்நுட்ப நவீனமயமாக்கும் திட்டத்தைத் தொடங்கியது. அடுத்த ஆண்டே அதன்

பராமரிப்பு செலவு மிகவும் அதிகம் என்பதை உணர்ந்து அடுத்து ஒரு விநியோக சங்கிலி மேலாண்மை மென்பொருளை மேம்படுத்த இரண்டாவது திட்டம் 3500 கோடி ரூபாய் முதலீட்டில் தொடங்கப்பட்டது. இந்த இரண்டு முயற்சிகளிலும் பிரச்சினைகள் வந்தன. 2002-ல் திவாலாவதாக கே-மார்ட் செய்த முடிவில் இந்த இரண்டு திட்டங்களுக்கும் பெரிய பங்கு உண்டு. பின்னர் இந்த நிறுவனம் 600-க்கும் மேற்பட்ட கடைகளை மூடி, 67,000 ஊழியர்களை வேலையை விட்டு நீக்கி சியர்ஸ் நிறுவனத்துடன் இணைந்தது.

- பாஃக்ஸ் மேயர் அமெரிக்காவில் நான்காவது பெரிய மருந்துகள் விநியோக நிறுவனம். 30,000 கோடி ரூபாய் மதிப்பு. 1993-ல் செயல்திறனை அதிகரிக்க ஒரு ERP அமைப்பும் மற்றும் தானியங்கி முறையில் கிடங்கும் அமைக்க 200 கோடி ரூபாய் முதலீட்டில் திட்டங்கள் தொடங்கப்பட்டன.

இதன் நேரடி விளைவாக 1996-ல் நிறுவனமே திவாலானது.

திட்ட மேலாண்மை அல்லது நிரலாக்கம் சரியில்லையா என்று நீங்கள் கேட்கலாம். நியாயமான கேள்வி. ஆற்றலுடையவர்கள்தான், மிகவும் ஈடுபாட்டுடன் வேலை செய்தார்கள். ஆனால் அனைவரும் சிறந்த முயற்சிகள் செய்தபோதிலும் மென்பொருள் உருவாக்கும் திட்டங்கள் ஏதோ காரணத்தினால் அடிக்கடி பிரச்சினையில் மூழ்குகின்றன.

நீங்கள் இந்த பிரச்சினைக்கு தீர்வு எங்கே இருந்து வந்திருக்கும் என்று நினைக்கிறீர்கள்? பிரம்மாண்டமான தோல்விகளில் இருந்து பாடம் கற்ற பெரும் நிறுவனங்கள் அல்லது அரசுத் துறைகள் அல்லது இது போன்ற பரந்த பிரச்சினைகளை பகுப்பாய்வு செய்யும் சிறந்த மேலாண்மைக் கல்வி நிறுவனங்களில் இருந்து என்றுதானே? இல்லை. இதற்கான தீர்வு சில எதிர்பாராத வேறு இடங்களிலிருந்து வந்தது.

என்ன தீர்வு, எங்கே இருந்து வந்தது என்று பார்க்கும் முன் நாம் மென்பொருள் உருவாக்கும் திட்டங்கள், குறிப்பாக பெரிய திட்டங்கள் எப்படி நிர்வகிக்கப்பட்டன, இன்றும் நிர்வகிக்கப்படுகின்றன என்பதை இந்த ''மென்பொருள் உருவாக்கும் விந்தையுலகம்'' தொடரில் ஆராய்வோம்.

உங்களுக்குத் தெரிந்த எந்த மென்பொருள் திட்டத்திலாவது பிரச்சினைகள் வந்ததுண்டா? கீழே உள்ள கருத்துப்பெட்டி மூலம் உங்கள் எண்ணங்களைப் பகிர்ந்து கொள்ளுங்கள். மென்பொருள் உருவாக்கும் திட்ட அணுகுமுறைகளில் என்ன குறைபாடு மற்றும் வித்தியாசமாகமாகவும், சிறப்பாகவும் எப்படி செய்யலாம் என்று என் கூட சேர்ந்து ஆய்வு செய்ய வாருங்கள்.

நன்றி,

இரா. அசோகன்

Ashok Ramachandran,
ashokramach@gmail.com

2. மென்பொருள் தேவைகள் தெரிவது என்பது மூடுபனியில் நடப்பது போன்றது!

பொதுவாக ஒரு மென்பொருள் உருவாக்கும் திட்டத்தில் பின்வரும் முக்கிய கட்டங்கள் உண்டு:

1. *தேவைப்பட்டியல் திரட்டுதல்*
2. *வடிவமைத்தல்*
3. *நிரலாக்கல்*
4. *சோதித்தல்*
5. *நிறுவுதல்*

மென்பொருள் திட்டங்களில் பிரச்சினை முதல் கட்டத்திலேயே தொடங்குகிறது. முதல் கோணல் முற்றிலும் கோணல் என்ற பழமொழி இதற்கு முற்றிலும் பொருந்தும்.

கணினி தகவல் அமைப்புகள் இதழில் வெளியான ஆராய்ச்சியின்படி 90% பெரிய மென்பொருள் திட்டங்களில் தோல்விக்கு மூலகாரணம் மோசமான தேவைகள் பட்டியல் திரட்டுதல்தான்.

இம்மாதிரி மோசமான தேவைகள் திரட்டுதலின் முக்கிய விளைவு scope creep. PMBOK (Project Management Body of Knowledge) என்பது சுமார் 600 பக்கங்கள் கொண்ட திட்ட மேலாளர்கள் அடிக்கடி எடுத்து படித்துப் பார்க்கும் பஞ்சாங்கம். PMBOK படி, scope creep-ன் வரையறை என்ன என்று பார்ப்போம். scope creep என்பது வாடிக்கையாளரின் எழுத்துமூல ஒப்புதல் இல்லாமலும் திட்டக்கெடு, செலவுகள், மற்றும் குழு உறுப்பினர்களின் வேலைச்சுமை

மீது ஆகும் விளைவுகள் பற்றி ஆராயாமலும் மென்பொருளில் அம்சங்களை மேலும் மேலும் ஏற்பதேயாகும்.

திட்டத்தை ஆரம்பித்த பின் எந்த மாற்றங்களும் செய்யக்கூடாது என்று சொல்லவில்லை. வேலை செய்யத் தொடங்கியபின் மாற்றங்கள் செய்தால் செய்த வேலையை மாற்றி அதிக செலவில் மறுவேலை நிறைய செய்ய வேண்டி வரும். எனவே, மிகவும் அவசிய மாற்றங்கள் மட்டுமே கேட்க வேண்டும். முறையான வழி மூலம் கேட்க வேண்டும். மற்றும் திட்டக் குழுவிடம் இம்மாதிரி கோரிக்கைகளால் திட்டக்கெடு, செலவுகள், மற்றும் குழு உறுப்பினர்களின் வேலைச்சுமை மீது ஆகும் விளைவுகளை ஆராய்ந்து வாடிக்கையாளருக்கு மீண்டும் மதிப்பீடுகள் வழங்க ஒரு செயல்முறை இருக்க வேண்டும். இந்தக் கூடுதல் செலவையும், திட்டக்கெடுவில் தாமதத்தையும் வாடிக்கையாளர் ஏற்றுக்கொண்டால், திட்ட

மேலாளர் அதன்படி திட்டத்தைத் திருத்தி அமைக்க வேண்டும்.

ஆனால் நடப்பதைப் பார்த்தால் இதெல்லாம் கனவுக்காட்சி போலில்லை?

ஆலோசனை நிறுவனம் Standish Group-ன் 2004 அறிக்கையின்படி தோல்வியுற்ற திட்டங்களில் 80% தேவைப்பட்டியல் திரட்டுதல் பிரச்சினையும் அதனால் வந்த scope creep-ம் தான்.

Computerworld செய்தி இதழ் எடுத்த கருத்துக் கணிப்பில் 160 தகவல் தொழில்நுட்ப நிபுணர்களில் 80% நபர்கள் scope creep "எப்போதும்" அல்லது "அடிக்கடி" வந்துவிடுகிறது என்று குறிப்பிட்டுள்ளார்கள்.

அவர்கள் கொடுத்த காரணங்கள் (பல தேர்வு):

- 44% மோசமான தேவைகள் பட்டியல்.

- *36% புதிய செயலியில் பயனர்களுக்குப் பரிச்சயமில்லை.*

- *28% திட்டங்கள் நீண்ட காலம் எடுத்ததால் தேவைகளே மாறிவிட்டன.*

- *22% பயனர் எதிர்பார்ப்பை தயார் செய்யத் தவறியது.*

- *19% ஆரம்ப கட்டங்களில் பயனர்களை ஈடுபடுத்தத் தவறியது.*

பயனர்களிடமிருந்து நல்ல உள்ளீடு இருந்தாலும் அவற்றை நன்கு பகுப்பாய்வு செய்தாலும் ஒன்று மட்டும் நிச்சயம். பயனர்கள் தங்களுக்கு என்ன வேண்டும் என்று தெரியும் என்று நினைக்கிறார்கள். ஆனால் அவர்கள் செயலியைப் பார்த்து பயன்படுத்தத் தொடங்கும்போதுதான் தங்களுடைய தேவை உண்மையில் என்ன என்று புரிந்து கொள்ள ஆரம்பிக்கிறார்கள். மென்பொருள் தேவைகள்

தெரிவது என்பது மூடுபனியில் நடப்பது போன்றது. தூரத்திலிருந்து பார்த்தால் மங்கலாகத்தான் தெரியும். அருகில் நெருங்க நெருங்க மிகத் துல்லியமாகத் தெரியத் தொடங்கும்.



அவர்கள் முன்னர் பயன்படுத்தாத ஒரு புதிய செயலி உருவாக்கும் போது இது குறிப்பாகப் பொருந்தும். வரைபட பயனர் இடைமுகப்பு (GUI) இருந்தால் இன்னும் மிக்கப் பொருந்தும். பெரும்பாலும் எல்லா மென்பொருள் திட்டங்களும் இந்த இரண்டில்தானே அடங்குகின்றன?

பல மென்பொருள் திட்டங்களில் நாம் எதிர்கொள்கின்ற இக்கட்டான நிலைமை இதுதான். புதிதாகத் தெரியவந்த தேவைகளைப் புறக்கணித்து விட்டு ஒப்பந்த ஆவணங்கள் படி செய்து பயனர்களுக்கு மிகத் தேவையான அம்சங்களை விட்டு விடுவதா? அல்லது தேவையான மாற்றங்கள் செய்து திட்டத்தின் செலவும் அதிகமாகி திட்டக்கெடுவில் முடிக்கவும் முடியாமல் திணறுவதா?

நான் திட்டத்தின் முதல் கட்டத்தில் பிரச்சினை தொடங்குகிறது என்றா சொன்னேன்? உண்மையில் திட்டம் துவங்கும் முன்னரே, திட்டத்துக்கு ஒப்புதல் பெற மதிப்பீடு

தயாரிப்பதிலேயே சிக்கல் ஆரம்பம். அது எப்படி என்று அடுத்த கட்டுரையில் காண்போம்.

scope creep-க்கு தமிழாக்கம் தேவை. *Feature creep, scope change* என்றும் சொல்கிறார்கள். இதோ இரண்டு முயற்சிகள்:

நகரும் குறியிலக்கு: *'moving target'*-க்கு அருகில் வருகிறது.

ஊதிப் பெருக்கும் செயற்பரப்பு: *Ballooning scope.* இது *'creep'* ஒத்த குறையான உட்பொருளை வழங்குகிறது. ஆனால், சிறிது நீளமாக இல்லை?

கீழே உள்ள கருத்துப்பெட்டியில் உங்கள் பரிந்துரைகளைப் பகிர்ந்து கொள்ளுங்கள். குழுச் சிந்திப்பில் பொருத்தமானதொரு சொற்றொடர் உருவாக்குவோம்.

3. எருமை மாட்டைத் தண்ணீரில் போட்டு விலை பேசுவது போல!

என் தந்தை ஒரு நெற்பயிர் விவசாயி. ஏதாவது முழுப் பரிமாணம் தெரியாத விஷயங்களைப் பற்றிப் பேசும்போது, "எருமை மாட்டைத் தண்ணீரில் போட்டு விலை பேசுவது போல" என்று உபமானம் கூறுவார். பெரும்பாலான மென்பொருள் மதிப்பீடும், பேரப் பேச்சும் எருமை மாட்டைத் தண்ணீரில் போட்டு விலை பேசுவது போலத்தான் நடக்கிறது. நீங்கள் ஒரு விற்பனையாளராக இருந்தால் அந்த திட்டத்தை மதிப்பீடு செய்தால்தான் நீங்கள் ஒரு விலை சொல்ல முடியும். நீங்கள் வாடிக்கையாளராக இருந்தால் இத்திட்டத்துக்கு ஆகும் முதலீடு என்ன, வருமானம் என்ன என்று பார்த்துத்தானே திட்டத்தைத் தொடருவதா வேண்டாமா என்று ஒரு முடிவு எடுப்பீர்கள்?

ஒரு கட்டிடத்தின் வடிவம், மாடிகள், அறைகள் எண்ணிக்கை மற்றும் வேறு பல மாறுபட்டாலும், பரவலாக இவைகளின் மத்தியில் ஒரு அலகு பொதுவாக உள்ளது. அது சதுர அடியில் கட்டிடத்தின் பரப்பளவு. வடிவமைப்பு செய்யும் முன்னரே ஒரு குறிப்பிட்ட இடத்தில் ஒரு குறிப்பிட்ட தரத்தில் கட்ட ஒரு சதுர அடிக்கு இன்னது செலவாகும் என்று தோராயமான மதிப்பீடு கட்டிடப் பொறியாளர்களால் தர முடியும். நாம் ஒரு கட்டிடத்தைக் கட்டுவது போல் மென்பொருள் திட்டங்களை மேலாண்மை செய்யப் போகிறோம் என்றால் நமக்கு ஒரு பொதுவான அலகு தேவை.


150
PROPOSED ADDITION

மென்பொருளுக்கு வெவ்வேறு பொது அலகுகள் முயற்சி செய்யப்பட்டன. அவற்றில் முதலில் வந்தது நிரலின் மூல வரிகள் SLOC (source lines of code). இது திட்டத்தின் மூல நிரலில் ஆவண வரிகளைத் தவிர்த்து மற்ற வரிகளின் எண்ணிக்கை. ஆனால் ஒரு மொழியில் 10 வரிகளில் எழுதுவதை மற்றொரு மொழியில் 15 வரிகளில் எழுத வேண்டியிருக்கலாம்.

மேலும் ஒரே மொழியில் திறமையான நிரலாளர்கள் குறைந்த வரிகள் கொண்டு அதே செயல்பாட்டை உருவாக்க முடியும். 1982-ம் ஆண்டு முதன்முதலில் செய்த ஆப்பிள் மேக் கணினி மென்பொருள் இறுதி வடிவம் கொடுக்கும் நேரம். திட்ட மேலாளர்கள் நிரலாளர்களை எழுதிய நிரல் வரிகளின் எண்ணிக்கையை ஒவ்வொரு வாரமும் சமர்ப்பிக்க வற்புறுத்தினர். பில் அட்கின்சன் இது அசட்டுத்தனம் என்று நினைத்தார். ஒரு வாரம் QuickDraw-வில் 2000 வரிகளைக் குறைத்து ஆறு மடங்கு வேகமாக ஓடச்செய்தார். அந்த வாரம்

அவர் படிவத்தில் “-2000” என்று எழுதினார். அதன் பின் மேலாளர்கள் படிவத்தை நிரப்ப வேண்டும் என்று கேட்பதை நிறுத்தி விட்டனர்! முயற்சியின் உள்ளீட்டு அளவுக்குப் பதிலாக முடித்த வேலையின் வெளியீட்டை அளவிடுவதே முக்கியம். செயல்பாட்டு புள்ளிகள் (Function Points) என்பது ஒரு செயலி ஒரு பயனருக்கு எவ்வளவு வேலைகள் செய்யப் பயன்படுகிறதோ அதன் அலகு. இதற்கு நன்கு நிறுவப்பட்ட தரங்கள், நல்ல குறிப்பு கையேடு, மற்றும் பயிற்சியாளர்களுக்கு சான்றிதழ் முதலியன உள்ளன. ஆனால் கடந்த 15 ஆண்டுகளில், செயல்பாட்டு புள்ளிகளை 10%-க்கும் குறைவான நிறுவனங்களும் திட்டங்களுமே பயன்படுத்துகின்றனர்.

மென்பொருளின் அளவை மதிப்பீடு செய்ய விற்பனையாளர் கணிசமான நேரமும் உழைப்பும் போட்டு ஈடுபட்டால் மட்டும் போதாது. வாடிக்கையாளரும் கணிசமான நேரமும் உழைப்பும் போட்டு ஈடுபட

வேண்டும். முக்கியமாக வணிக பிரச்சினை என்ன, அந்த பிரச்சினையை தீர்ப்பதற்கு என்ன மென்பொருள் செயலி வேலைக்கு ஆகும் என்ற முன்யோசனை உள்ள மூத்த நிர்வாகிகள் தங்கள் தேவைகளை வழங்க கணிசமான நேரத்தை ஒதுக்க வேண்டும். இது பெரும்பாலும் நடப்பதில்லை. ஆனால் பெரும்பாலான மேல்மட்ட நிர்வாகிகளோ ரொம்பவும் மெனக்கெடாமல் நேரமும் எடுக்காமல் மந்திரக்கோலை அசைத்து உடன் மதிப்பீடு தரவேண்டும் என்று எதிர் பார்க்கிறார்கள்.

இந்த காரணத்தால் மென்பொருள் உருவாக்கும் முயற்சி அளவை மதிப்பிடுவது பெரும்பாலும் நிபுணர்கள்தான் குத்துமதிப்பாகச் செய்கிறார்கள் என்று வெளியிடப்பட்ட ஆய்வுகள் கூறுகின்றன. ஆனால் நிபுணர்கள் வெகுசீக்கிரம் வேலையை முடித்துவிட முடியும் என்ற தளரா நம்பிக்கையுடனே மதிப்பீடுகள் செய்கின்றனர். மேலும் பல நிரலாளர்கள் இயலக் கூடிய மதிப்பீடுகள் செய்தால் தாங்கள்

சோம்பேறிகளாகவோ அல்லது திறனற்றவர்களாகவோ தோன்றலாம் என்று பயப்படுகிறார்கள் போலிருக்கிறது. அல்லது அவர்கள் கணக்கில், மென்பொருளை சோதிக்கவும், கண்காணிக்கவும், கட்டமைக்கவும், நிறுவவும், ஆவணப்படுத்தவும், பராமரிக்கவும் தேவைப்படும் உழைப்பையும் நேரத்தையும் எடுத்து கொள்வதில்லை போல் தெரிகிறது. நிரலாளர்களும் தொழில்நுட்ப நிபுணர்களும் திரும்பத் திரும்ப மென்பொருள் உருவாக்கத் தேவைப்படும் முயற்சியையும் நேரத்தையும் குறைத்தே மதிப்பீடு செய்கின்றனர்.

மேலும் தொழில்நுட்பம் மாறுவதால் பணித்திட்டத்துக்கு எழும் இடர்களை நிபுணர்கள் கணக்கில் எடுப்பதாகத் தெரியவில்லை. நிரல் எழுதத் தொடங்கிய பின்தான் நமக்கு வடிவமைப்பில் உள்ள தவறுகள் பெரும்பாலும் தெரிய வரும்.

இதைவிடக் கொடுமை என்னவென்றால் திட்ட இறுதியில் செயல்திறன் சோதனை தொடங்கிய பின் தெரிய வருவது.
எடுத்துக்காட்டாக ஆயிரக்கணக்கான பயனர்களுக்கு ஆதரவு தரவேண்டிய ஒரு ஊதிய திட்டம். வெறும் பத்து பயனர்கள் கொண்டு சோதனை செய்தபோது ஆமை வேகத்தில்தான் நகர்ந்தது. முழுத் திட்ட வேலையும் செய்த பின், இரண்டு ஆண்டுகளாக வாடிக்கையாளர் மற்றும் விற்பனையாளர் இருவருக்கும் பெரும் இழப்பு ஆன பின், ரத்து செய்யப்பட்டது.
தொழில்நுட்பங்கள் உயர்வதும் தாழ்வதும், இயக்கமுறைமை வெளியீடுகள், பாதுகாப்பு பிரச்சினைகள், சேவையளிக்கும் நிறுவனங்கள் வாங்கப்படுவதும் திவாலாவதும் முதலாக திட்டத்துக்கு இடராக வரக்கூடிய தொழில்நுட்ப நிகழ்ச்சிகள் அடிக்கடி நடந்துகொண்டுதானே இருக்கின்றன?

மேலும் மென்பொருள் நிரல்களின் ஒட்டுமொத்த அளவு சம்பந்தப்பட்டவர்களுக்கு

மிகவும் புரியாத புதிராகவே இருக்கிறது. எடுத்துக்காட்டாக நீங்கள் ஒரு 6 அறைகள் உள்ள 2 மாடி வீடு கட்ட திட்டம் போடுகிறீர்கள் என்று வைத்துக்கொள்வோம். நீங்கள், உங்கள் கட்டிடக் கலைஞர், கட்டிடப் பொறியாளர், ஒப்பந்தக்காரரர், ஆசாரி, கொத்தனார், சித்தாள் முதல் உங்கள் குடும்பத்தார் வரை எல்லோருக்கும் ஓரளவுக்கு அதே உத்தேச அளவுதான் மனதிலிருக்கும். ஆனால் மென்பொருள் உருவமில்லாதது, எளிதில் உணர்ந்தறிய முடியாதது.

மேலும் இதை சிக்கலாக்கும்படி, ஒரு ஒற்றை திறமையான நிரலாளர் இராப்பகலாக உழைத்து மிகவும் விரைவில் ஒரு மிகப் பெரிய முழு செயலியை உருவாக்க முடியும். ஒரு தொழிலாளி தனியாக ஒரு முழுக் கட்டடத்தையும் கட்ட முயலுவதை இத்துடன் ஒப்பிட்டுப் பாருங்கள். எவரும் 6 மாதங்களில் ஒரு பிரம்மாண்டமான கட்டிடத்தை கட்டச் சொல்லி ஒரு சிறிய ஒப்பந்ததாரரிடம் கேட்க மாட்டார்கள். ஆனால்

இதன் விளைவாக வாடிக்கையாளர்கள் அடிக்கடி மென்பொருள் நிறுவனங்களை பிரம்மாண்டமான கட்டிடத்துக்குச் சமமான மென்பொருள் பணியை செய்ய எதிர்பார்க்கிறார்கள்.

ஆக மென்பொருள் திட்டங்களை மதிப்பீடு செய்யும் போதும், முதலீடு செய்ய முடிவெடுக்கும் போதும், பேரம் பேசும் போதும், ஒப்பந்தம் செய்யும் போதும் எருமை மாட்டைத் தண்ணீரில் போட்டு விலை பேசுவது மட்டும் அல்ல, அது எருமை மாடா, காண்டாமிருகமா, யானையா என்று கூட சரியாகத் தெரியாமல்தான் பேசிக்கொண்டிருக்கிறோம்!

4. திட்டம் 40% முடிந்தும் நிரல் ஒரு வரி கூட இல்லை ஆனால் ஆவணங்களோ ஒரு அடுக்கு!

நான் அப்போது அமெரிக்காவில் ஒரு மத்திய அரசாங்கத் துறையில் ஆலோசகராக இருந்தேன். கூடிப்பேச சென்றிருந்த திட்ட இயக்குனர் அப்பொழுதுதான் திரும்பி வந்திருந்தார், அதிர்ச்சியுடன் காணப்பட்டார். அந்த நிகழ்ச்சியில் நடந்தது பற்றி யாரிடமாவது சொல்லா விட்டால் மண்டை வெடித்து விடும் நிலையில் இருந்தார். ஒப்பந்தக்காரர்கள் உருவாக்கிய மென்பொருளைக் காட்டி, விளக்கி, கொண்டு சேர்க்க வந்திருந்தனராம். ஒப்பந்தத்திலுள்ள தேவைப் பட்டியல்படி மூன்று ஆண்டுகளாகக் கடுமையாக உழைத்து

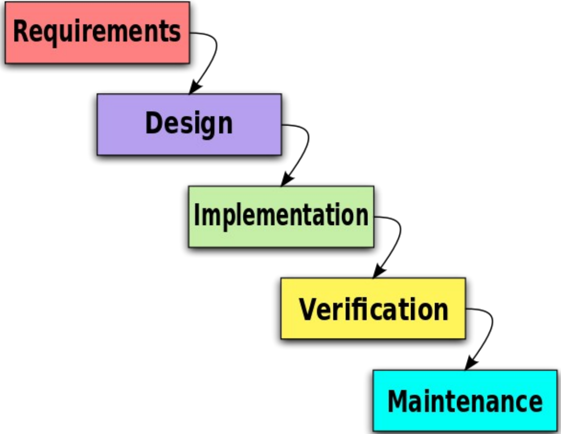
உருவாக்கியதென்று கூறினார்களாம். செயல்முறைக்காட்சியின் இறுதியில் வாடிக்கையாளர் அணி, “இதுவல்ல நாங்கள் கேட்டது” என்று கூறி அந்த முழு விநியோகத்தையும் அறவே நிராகரித்து விட்டனராம்.

மென்பொருள் திட்டங்களில் இது போன்ற பெரும் நெருக்கடிகள் ஏன் ஏற்படுகின்றன என்று புரிந்து கொள்ள நாம் **அருவி செயல்முறை *(Waterfall Method)*** என்றால் என்ன, அது எங்கிருந்து வந்தது, எப்படி வேலை செய்கிறதென்று தெரிந்து கொள்ள வேண்டும். உற்பத்தி மற்றும் கட்டுமான தொழில்துறை திட்டங்களில் இந்த செயல்முறை உருவானது. இத் தொழில்துறைகளில் வேலையை ஒரு முறை செய்த பிறகு அதை மாற்றியமைப்பது சில சமயம் சாத்தியமேயில்லை. மற்ற சமயங்களில் சிறு மாற்றங்கள் கூட அதிக நேரம் எடுக்கும், அநியாய செலவு வைக்கும். எடுத்துக்காட்டாக, கான்கிரீட் போட்ட பின்னர் வடிவமைப்பை

மாற்ற விரும்பினால் எப்படி செய்வதென்று எண்ணிப் பாருங்கள்.

இப்படி இருக்கும்போது வடிவமைப்பை முற்றிலும் திருப்தியாக முடித்து வாடிக்கையாளரிடம் கையெழுத்தும் வாங்கிய பின் கட்டுமான வேலையைத் தொடங்குவீர்களா? அல்லது கட்ட ஆரம்பித்த பின் திரும்பிச் சென்று வடிவமைப்பைக் கொஞ்சம் அழகு படுத்துவீர்களா?

மென்பொருள் உருவாக்கத்தில் இதைத்தான் நாம் **அருவி செயல்முறை** என்கிறோம். ஏனென்றால் இதில் முன்னேற்றம் ஒவ்வொரு கட்டத்திலும் கீழ்நோக்கி படிப்படியாக அருவி போன்று பாய்கிறது. ஒவ்வொரு கட்டத்தையும் ஒரு முறைக்கு இரண்டு முறை சரிபார்த்து விட்டுத்தான் அடுத்த கட்டத்துக்குச் செல்வீர்கள். முன்னால் முடித்த கட்டத்துக்கு ஒரு போதும் திரும்பிச் செல்ல மாட்டீர்கள்.



commons.wikimedia.org/wiki/File:Waterfall_model_%281%29.svg

மென்பொருள் திட்டங்கள் ஆரம்பித்த காலத்தில் திட்ட மேலாளர்களும் மூத்த நிர்வாகிகளும் இந்த முறையில் மற்ற பல திட்டப்பணிகளை நிர்வகித்து அனுபவம் பெற்றிருந்தனர். மேலும்,

இவர்களுக்கு பரிச்சயமான, வழக்கமாகப் பயன்படுத்தும் திட்ட மேலாண்மைக்கான மென்பொருட்களில் இந்தச் செயல்முறை உள்ளிடப்பட்டு இருந்தது. ஆகவே இந்தச் செயல்முறையையே இயல்பாகப் பின்பற்றினர்.

இது போதாதென்று பல ஆண்டுகளுக்கு முன் செய்த ஒரு ஆராய்ச்சியின்படி மென்பொருள் உருவாக்கும் போது ஆரம்ப கட்டங்களில் ஒரு வழுவை நீக்குவது எளிது, ஆகையினால் அதற்கான செலவும் குறைவு. அதே வழுவை பின்னர் வரும் கட்டங்களில் நீக்க 50 முதல் 200 மடங்கு அதிக செலவாகலாம் என்ற முடிவு வெளி வந்தது. ஆனால் இந்த ஆராய்ச்சி FORTRAN மற்றும் COBOL மொழிகளில் நிரலாக்கம் எழுதி, துளையிடப்பட்ட அட்டைகளில் உள்ளீடு செய்த காலத்தில் நடந்தது. இந்த மொழிகளை அப்போது தொகுப்பதும், சோதனை செய்வதும் மிக மெதுவாகத்தான் நடக்கும், அதற்கான செலவும் அதிகம். எனினும் இது எக்காலத்திலும் எம்மொழிக்கும் பொருந்தும் என்று பரவலாக

நம்பப்பட்டது. இதன் காரணமாக முன்னால் உள்ள கட்டங்களில் வழுக்கள் ஏற்டாமல் இருக்க பகீரதப் பிரயத்தனம் செய்தனர்.

மேற்கண்ட காரணங்களாலோ, அல்லது அரசாங்க வகை எழுத்துமூல ஒப்பந்தங்கள் செயல்படுத்துவது எளிதாக இருக்கும் என்றோ மென்பொருள் கொள்முதலுக்கு அருவி செயல்முறையையே பின்பற்றுவதென அமெரிக்க அரசாங்க பாதுகாப்புத் துறை முடிவு செய்தது. மற்ற அரசாங்கத் துறைகளும், பெரிய நிறுவனங்களும் பின் தொடர்ந்தன. இதுவே மேலோங்கிய செயல்முறையாக நிலைபெற்றது.

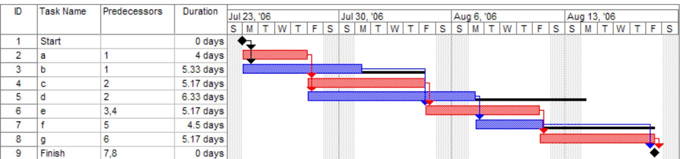
பொதுவாக நடைமுறையில், அருவி முறை எப்படி செயல் பட்டது என்று பார்ப்போம். இதன் முதல் படியாக தேவைகள் பட்டியல் தயார் செய்து வாடிக்கையாளர்கள் அதில் கையெழுத்திட்டு ஒப்புதல் கொடுத்து விட்டார்கள் என்று வைத்துக்கொள்வோம். திட்ட மேலாளர்கள் தொழில்நுட்ப குழுவுடன்

அமர்ந்து முழுத் திட்டத்தையும் முடிக்க என்னென்ன வேலைகள் செய்ய வேண்டும், அவை ஒவ்வொன்றுக்கும் எவ்வளவு நேரம் எடுக்குமென்ற பட்டியல் தயார் செய்வர். இதை பணிக் கூறுகள் அமைப்பு *(Work breakdown structure)* என்று சொல்கிறோம். இத்துடன் இப் பணிகளின் சார்புநிலையையும் *(dependency)* குறிப்பெடுக்க வேண்டும். அதாவது ஒரு பணியை ஆரம்பிக்கும் முன் அது சார்ந்திருக்கும் எந்தெந்த பணிகள் முடிந்திருக்க வேண்டும் என்பது. எடுத்துக்காட்டாக, சுவற்றில் வெள்ளை அடிக்கும் முன் சிமெண்ட் பூசிக் காய்ந்திருக்க வேண்டும் அல்லவா? என்னதான் அவசரம் என்றாலும் இதை மாற்றிச் செய்ய முடியாதே. அடுத்து பணியாளர்கள் ஒதுக்கீடும் *(resource allocation)* செய்வர்.

மொத்த திட்டத்துக்கும் இவ்வாறான எல்லா ஆய்வுகளையும் முடித்து திட்ட மேலாண்மை மென்பொருளில் உள்ளிட்ட பின், திட்டத்தை எப்பொழுது முடிக்க இயலும் என்று தெரிய

வரும். இன்னும் விரைவில் முடிக்க வேண்டுமென்றால் பணிகளை கூடியவரை மாற்றியமைத்துப் பார்க்கலாம். அப்படியும் ஆகாவிட்டால் மேலும் பணியாளர்களை ஒதுக்கீடு செய்ய வேண்டியிருக்கும்.

வேலை நடக்கும் போது காண்ட் *(Gantt)* வரைபடம் ஒவ்வொரு பணிக்கும் தொடங்கும் தேதியையும் முடிக்கும் தேதியையும் காட்டும். மற்றும் யார் எந்த வேலையை செய்ய வேண்டும், அந்த வேலையை ஆரம்பிக்கும் முன் மற்ற எந்த வேலைகள் முடிந்திருக்க வேண்டும் என்றும் காட்டும். பணிகளின் சதவீத-முழுமையை நிழல் கோடிட்டுக் காட்டும். தற்போதைய அட்டவணை நிலையை “இன்று”



ID	Task Name	Predecessors	Duration
1	Start		0 days
2	a	1	4 days
3	b	1	5.33 days
4	c	2	5.17 days
5	d	2	6.33 days
6	e	3,4	5.17 days
7	f	5	4.5 days
8	g	6	5.17 days
9	Finish	7,8	0 days

என ஒரு செங்குத்து கோடிட்டுக் காட்டும்.

en.wikipedia.org/wiki/File:Pert_example_gantt_chart.gif

ஒரு வாரத்தில் நூற்றுக்கணக்கான பணிகள் நடக்கும்போது ஒவ்வொன்றாக ஆச்சா, ஆச்சா என்று பார்க்க வேண்டியதில்லை. எந்த 4 அல்லது 5 பணிகள் ஆகவில்லையோ அதைமட்டும் ஏன் ஆகவில்லை, என்ன செய்து விட்டதைப்பிடிக்கலாம் என்று பார்க்கவேண்டும். மேலும் அம்மாதிரி பணிகள் திரும்பவும் தாமதம் ஆவதை எப்படித் தவிர்க்கலாம் என்று பார்ப்பதும் மிக முக்கியம். இதை விதிவிலக்குகளை கவனிக்கும் மேலாண்மை (Management by Exception) என்று கூறுகிறோம்.

அருவி செயல்முறையில் ஒரு திட்டம் ஆரம்பித்த பின் மாற்றங்கள் செய்வது கடினம். அறவே முடியாதென்று சொல்லவில்லை. ஆனால் சில நேரங்களில் சிறிய மாற்றங்கள் கூட தொடர் விளைவுகளை ஏற்படுத்தும். இதைக் கையாளுவதற்கு திட்ட மேலாளர் பல

உறுப்பினர்களைக் கலந்தாலோசிக்க வேண்டி வரலாம். அந்த மாற்றங்கள் செய்வதற்கான செலவையும் எளிதில் மதிப்பிட இயலாது. மேலும் மாற்றங்களின் செயற்பரப்பை பொருத்து பல மட்டங்களில் அனுமதி வாங்க வேண்டியிருக்கலாம். ஆக அருவி செயல்முறை சீட்டுக்கட்டு வீடு போன்றது. கட்டிய பின் கீழேயுள்ள ஒரு சீட்டை மட்டும் உருவி மாற்ற வேண்டுமென்றால் அது எளிதில் ஆகாது.

அருவி செயல்முறை எதற்குமே பயன்படாது என்று சொல்லவில்லை. நீங்கள் விண்வெளித் திட்டத்துக்கு மென்பொருள் எழுதப்போகிறீர்கள் என்றால், அருவி செயல்முறையே அதற்கு உகந்தது. நாம் அன்றாடம் செய்யும் வணிக, இணைய, அலைபேசி செயலிகள் போன்ற மென்பொருள் திட்டங்களுக்கு அது ஏற்றது அல்ல.

அருவி செயல்முறையின் முதல் இரண்டு கட்டங்களுக்கு, அதாவது தேவைப்பட்டியல் திரட்டுதல் மற்றும் வடிவமைத்தல், மொத்த

திட்ட நேரத்தில் 20 முதல் 40% நேரம் எடுக்கும். எடுத்துக்காட்டாக ஒரு திட்டப்பணிக்கு மொத்தம் ஒரு வருடம் ஆகும் என்று வைத்துக்கொள்வோம். முதல் 3 முதல் 5 மாதங்கள் முடிந்த பின் ஒப்பந்தப்படி வாடிக்கையாளருக்கு ஒரு அடுக்கு ஆவணங்கள் அனுப்பி இருப்பீர்கள். வாடிக்கையாளரும் 40 முதல் 60% பண வழங்கீடு செய்திருப்பார்கள். ஆனால் ஒற்றை வரி கூட நிரல் எழுதியிருக்க மாட்டீர்கள். அப்படி ஒருக்கால் நீங்கள் ஏதாவது எழுதியிருந்தாலும் வாடிக்கையாளர் பார்த்திருக்க மாட்டார்கள், பார்க்கவும் வழியில்லை!

5. ஆவணங்களைக் குறைத்து மென்பொருளை தேவைக்குத் தக அமைப்பதை எளிதாக்குங்கள்!

ஒருவேளை நீங்கள் அரசியல் கட்சிகள் மட்டும்தான் கொள்கை விளக்க அறிக்கைகளை தேர்தல் நேரத்தில் வெளியிடுவார்கள் என்று நினைத்தீர்களா என்ன? 2001-ம் ஆண்டு குளிர்காலத்தில் யூடா மாகாணத்தின் ஸ்னோபேர்ட் பனிச்சறுக்கு மையத்தில் 17 மென்பொருள் உருவாக்குபவர்கள் கூடினர். இவர்கள் யாவரும் வெவ்வேறு மென்பொருள் செயல்முறைகளை (Extreme Programming or XP, Scrum, DSDM, Adaptive Software Development, Crystal, Feature-Driven Development, Pragmatic Programming, and others) உருவாக்கி ஊக்குவித்து வந்தனர்.

ஆனாலும் பொதுவாக இவர்கள் பயன்படுத்திய வழிமுறைகள் பேச்சுவழக்கில் “இலகுரக வழிமுறைகள்” என்று குறிப்பிடப்பட்டன. ஏனென்றால் அருவி வழிமுறை போலில்லாமல் இவர்களின் வழிமுறைகள் ஆவணங்களைக் குறைத்து நிரலாக்கத்தில் அதிக முக்கியத்துவம் வைத்தன. இவர்கள் சந்தித்து, கூடிப்பேசி, தங்கள் வழிமுறைகளில் பகிர்ந்த நம்பிக்கைகளை வெளிக்கொணர்ந்து, ஒருமுகப்படுத்த முயற்சி செய்தனர். இந்த முயற்சியின் முடிவில் மென்பொருள் உருவாக்குவதற்கான தங்களுடைய கொள்கை விளக்க அறிக்கையையும் (*Agile Manifesto*) அத்துடன் மென்பொருளுக்கான கோட்பாடுகளையும் வெளியிட்டனர். இந்த கொள்கைகளும் கோட்பாடுகளும் ஒவ்வொரு விதத்திலும் அருவி செயல்முறைக்கு எதிர்மாறாக இருந்தன.

ஆவணங்கள் பற்றி

- விரிவான ஆவணங்களை விட வேலை செய்யும் மென்பொருளே முக்கியமானது. திட்டத்தில் வேலை எவ்வளவு முடிந்தது என்பதற்கு முதன்மையான அளவு மென்பொருள்தான். ஆகவே நாங்கள் இடைவிடாமல் மதிப்புமிக்க மென்பொருளை உருவாக்கி வெளியீடு செய்வதன் மூலம் வாடிக்கையாளரை திருப்தி செய்வதே முக்கியம் என்று நம்புகிறோம்.

மாற்றங்கள் பற்றி

- வாடிக்கையாளர்களால் அவர்களுக்கு தேவைப்படும் மென்பொருள் தேவைகளை பட்டியலிடுவது சாத்தியமற்றது. அவர்கள் ஒருக்கால் இதைச் செய்தாலும், மென்பொருள் வெளியீடு செய்வதற்குள் அந்த தேவைகளில் மாற்றம் ஏற்படலாம்.

- வாடிக்கையாளர் தேவைகளும், தொழில்நுட்பமும் மாறிவிட்ட பின் கண்ணைமூடிக்கொண்டு முன் போட்ட திட்டத்தையே பின்பற்றுவது விவேகமல்ல.

- திட்டத்தின் கடைசி கட்டங்களில் கூட மாறிவரும் தேவைகளை வரவேற்கிறோம். மாற்றங்களை கட்டுப்படுத்தி பயன்படுத்துவதன் மூலம் வாடிக்கையாளருக்கு சந்தைப் போட்டியில் அனுகூலம் செய்வோம்.

திட்ட இறுதியில் வெளியீடு செய்வது பற்றி

- வணிகத்துறையினரும் நிரலாளர்களும் தினந்தோரும் இணைந்து வேலை செய்ய வேண்டும். அருவி செயல்முறையில் ஒப்பந்தக்காரரர் கொள்முதல் ஆணையை வாங்கிக்கொண்டு சென்று வேலையை முடித்தபின் முழுமையான நிரலை

கொண்டுவந்து சேர்ப்பதை இத்துடன் ஒப்பிட்டுப் பாருங்கள்.

•மென்பொருளை உடனுக்குடன் வெளியீடு செய்வதன் மூலம் உங்கள் திட்டத்தின் தற்போதைய நிலை தெளிவாவது மட்டுமல்லாமல் பங்குதாரர்கள் உடன் பின்னூட்டம் வழங்கவும் இயலும்.

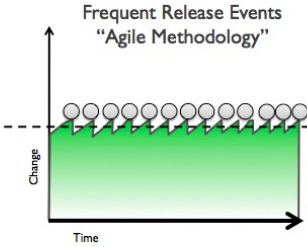
வேலை செய்யும் அணிகள் பற்றி

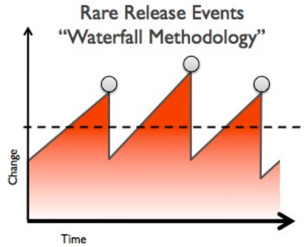
•குழு உறுப்பினர்களின் திறமையும், அவர்கள் ஒன்றாக இணைந்து வேலை செய்வதும் மிக முக்கியம். அது சரியாக நடக்கவில்லை என்றால் சிறந்த கருவிகளாலும் செயல்முறைகளாலும் எந்தப் பயனும் ஏற்படாது.

•சுயமாக வேலையை பங்கிட்டுச்செய்யும் அணிகளே சிறந்த கட்டமைப்புகளையும், தேவைகளையும், வடிவமைப்புகளையும் உருவாக்குகின்றனர்.

- மென்பொருள் வழிமுறைகளில் மேம்பாடு என்பது தொடர்ந்து செய்ய வேண்டிய முயற்சி ஆகும். செயல் முடித்த பின் இன்னும் திறம்படசெய்வது எப்படி என்று ஆய்வு செய்து அணிகள் தங்கள் நடவடிக்கைகளை சரிப்படுத்திக் கொண்டே இருக்க வேண்டும்.

- இந்த செயல்முறைகள் நிலைத்திருக்கக்கூடிய வளர்ச்சியை ஊக்குவிக்கின்றன. பங்குதாரர்கள், நிரலாளர்கள், மற்றும் பயனர்கள் காலவரையின்றி ஒரு நிலையான வேகத்தை பராமரிக்க முடியும். அருவி செயல்முறையில் திட்டக் கெடுவில் முடிக்க செய்யும் மரண அணிவகுப்பை இத்துடன் ஒப்பிட்டுப் பாருங்கள். மரண அணிவகுப்பு என்பது கடைசி நிமிடத்தில் அவசர அவசரமாகச் செய்யும் களைக்க வைக்கும் அளவுக்கு மீறிய வேலையைக் குறிக்கிறது.

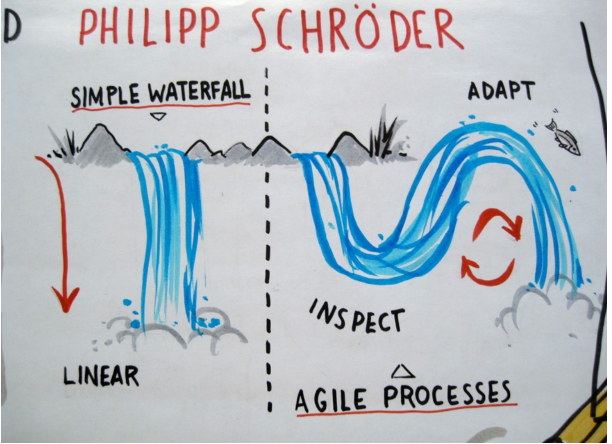




commons.wikimedia.org/wiki/File:Agile-vs-iterative-flow.jpg

இந்த *Agile Manifesto* மென்பொருள் உருவாக்கும் செயல்முறைகளை நிரந்தரமாக மாற்றியது. வணிக மற்றும் தொழில்நுட்ப உலகில் ஏற்றுக்கொள்ளப்பட்ட, ஆனால் மிகப் பிரச்சினையானதென்று உணர்ந்த, மென்பொருள் உருவாக்கும் அணுகுமுறையான அருவி செயல்முறை வேலைக்கு ஆகாது என்று

இந்த *Agile Manifesto* பிரகடனம் செய்தது. அருவி செயல்முறை பழுதானது. இம்முறையில் சந்தையிலும் தொழில் நுட்பங்களிலும் ஏற்படும் மாற்றங்களுக்கு ஏற்ப மென்பொருளை உடன் மாற்றுவது மிகக் கடினம். திட்டம் வெற்றியில் முடியுமா அல்லது தோல்வியில் முடியுமா என்பது கடைசி நிமிடம் வரை மதில் மேல் பூனையாகவே இருக்கிறது. இந்தக் காரணங்களால் அருவி செயல்முறை அடிப்படையிலேயே குறைபாடுகளுடையது.



farm8.staticflickr.com/7314/8746306184_502e6dcccf_o_d.jpg

திட்டத்தில் முன்னேற்றம் பற்றி வாடிக்கையாளருக்கு ஆவணங்கள் அனுப்புவது புரோநோட்டு எழுதித் தருவது போல்தான்

என்று கூறுகிறார் Crystal-ன் ஆலிஸ்டேர் காக்பர்ன் (Alistair Cockburn). புரோநோட்டு எழுதித் தருவதற்கு பதிலாக தவணை முறையில் பணமே செலுத்துங்கள். சோதனை செய்த, ஓடுகின்ற நிரலின் கூறுகளை நீங்கள் அடிக்கடி விநியோகம் செய்வதனால் திட்டத்தை குறித்த கெடுவில் முடிக்கும் சாத்தியக்கூறு அதிகமாகிறது.

இந்த கொள்கை விளக்க அறிக்கை நிரலாளர்களால் உருவாக்கப்பட்டது. Pragmatic Programming டேவிட் தாமஸ் (David Thomas) கூறுகிறார், “1980 மற்றும் 90-ல் இருந்த அவசியமற்ற, ஆன்மாவை அழிக்கும் நடை முறைகள் சிலவற்றிலிருந்து நிரலாளர்கள் உடைத்து வெளிவர இது உதவியது.”

மேற்குறிப்பிட்ட பல இலகுரக வழிமுறைகளில் திட்ட மேலாண்மைக்கு Scrum செயல்முறையும் பொறியியல் நடைமுறைகளுக்கு XP-ம் மேலோங்கின. தற்போதைய சிறந்த நடைமுறை இந்த இரண்டையும் ஒருங்கிணைத்தது. Scrum

செயல்முறையை உருவாக்கி, வெற்றிகரமாக செயல்படுத்தி, பிரபலப்படுத்தியவர்கள் ஜெஃப் சதர்லேண்ட் (Jeff Sutherland) மற்றும் கென் ஷ்வாபர் (Ken Schwaber). பொறியியல் நடைமுறையில் பல புதிய அம்சங்களை முயற்சி செய்து அவற்றின் செயல்திறனை நிரூபித்து நிலைநாட்டியவர் XP-ன் கென்ட் பெக் (Kent Beck). ஆகவே இவர்களையும் இவர்களின் பங்களிப்புகளையும் பற்றி அடுத்து வரும் கட்டுரைகளில் விவரமாகக் காண்போம்.

Agile-க்கு தமிழாக்கம் "தகவெளிமை" என்று கருத்தனின் சிந்தனைப் பட்டறையில் விளக்கமாக எழுதியுள்ளார். எனக்கும் இது சரியாகவே படுகிறது. நீங்கள் என்ன நினைக்கிறீர்கள்? கீழே உள்ள கருத்துப்பெட்டி மூலம் உங்கள் எண்ணங்களைப் பகிர்ந்து கொள்ளுங்கள்.

6. மென்பொருள் திட்டம் நிர்வகிக்க, போர் விமானத்தை தரையிறக்கப் பழகுங்கள்!

புதிய உற்பத்திப்பொருட்கள் கண்டுபிடிக்கும் ஒரு குழு எந்த மாதிரி அணுகுமுறை பயன்படுத்தலாம் என்று 1986-ல் டாகெயூச்சி மற்றும் நோனாகா ஒரு ஆய்வுக் கட்டுரை எழுதினர். வாகனம், நகலி மற்றும் அச்சுப்பொறி தயாரிப்பு நிறுவனங்களில் செய்த நேர் ஆய்வுகளின் அடிப்படையில் இதை எழுதினர். வழக்கமாகச் செய்யும் தொடர்நிலை அணுகுமுறைக்குப் பதிலாக Scrum* அணுகுமுறை இம்மாதிரி வேலைக்கு திறம்பட்டதாக இருக்கும் என்பது அவர்கள் பரிந்துரை. ரக்பி (Rugby) கால்பந்தாட்டத்தில் ஏதும் சிறிய தவறினால் ஆட்டம் நின்ற பின்

திரும்பத் தொடங்கும் முறையை *Scrum* என்று சொல்கிறார்கள். ஒருவருக்கொருவர் பந்தைப் போட்டு பிடித்து அணியினர் ஒட்டு மொத்தமாக முன்னேறிச் செல்ல முயற்சிப்பார்கள். வேகமாகச் செல்லவும் முடியும், எதிர் அணியின் ஆட்டம் மற்றும் எஞ்சியுள்ள நேரம் போன்ற தேவைகளுக்குத் தகுந்தாற்போல் அணுகுமுறையை மாற்றிக் கொள்ளவும் முடியும். சுயமாக வேலையைப் பங்கிட்டுச்செய்யும் அணிகளே இம்மாதிரி வேலைக்கு உகந்தவை.

1993-ல் ஜெஃப் சதர்லேண்ட் (Jeff Sutherland) ஈசல் நிறுவனத்தில் *Object Studio* என்ற புதிய மென்பொருள் செயலி தயாரிக்க முதன்மை பொறியாளராகச் சேர்ந்திருந்தார். இந்த கட்டுரையால் மன எழுச்சியுற்று திட்டம் நிர்வகிக்க *Scrum* அணுகுமுறையைச் செயல்படுத்த முயற்சித்தார்.

மற்றொரு ஆராய்ச்சிக் கட்டுரை போர்லாண்ட் நிறுவனத்தின் *Quattro Pro* செயலி

பற்றியது. அதற்கு எட்டு பேர் குழு முப்பத்தோரு மாதங்களில் ஒரு மில்லியன் வரிகள் C++ நிரல் எழுதியது. அதாவது ஒவ்வொரு குழு உறுப்பினரும் வாரத்துக்கு ஆயிரம் வரிகள் எழுதியுள்ளனர். இது நிரல் எழுதுவதில் ஒரு அதிவேக சாதனை. இதில் ஒரு தனிப்பட்ட அம்சம் என்னவென்றால் அந்தக் குழுவின் உறுப்பினர்கள் ஒவ்வொரு நாளும் கூடிப் பேசினர். அனைவரையும் ஒரே அறையில் கொண்டு வருவது முக்கியம். ஏனென்றால் எதுவும் சவாலான பிரச்சினை வந்தால் உடன் சுயமாக வேலையை பங்கிட்டுத் தீர்வு காண இயலும்.

இதன் அடிப்படையில்தான் அவர் தினசரி கூடிநின்று பேசல் ஆரம்பித்தார், "ஈசல் நிறுவனத்தில் ஒரு குறுவோட்டத்தில் (Sprint) வழக்கம்போல நான்கு வாரங்களுக்கான வேலை செய்வதென்று திட்டம் போட்டிருந்தோம். நாங்கள் தினசரி கூடிநின்று பேச ஆரம்பித்தோம். வேலை முழுவதையும் ஒரே வாரத்தில் முடித்து விட்டோம். 400% மேம்பாடு! அந்த முதல்

வெள்ளிக் கிழமை கூடிநின்று பேசும்போது குழு உறுப்பினர்கள் ஒருவரையொருவர் பார்த்துக்கொண்டு ‘அபாரம்’ என்றோம். அந்தக் கணத்தில்தான் எனக்கு ஏதோ புதிய பாதையில் நுழைகிறோம் என்று தோன்றியது.”

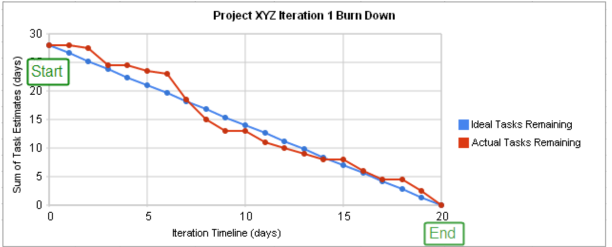
ஜெஃப் சதர்லேண்ட் அமெரிக்க விமானப்படையில் ஒரு போர் விமானியாக 11 ஆண்டுகள் வேலை செய்தவர். அவர் கூறுகிறார், “மென்பொருள் உருவாக்க திட்டத்தை மட்டும் விவரமாகத் தயார் செய்து விட்டால் போதும், எல்லாம் சொல்லி வைத்தாற்போல் அப்படியே நடக்கும் என்று எதிர்பார்க்கிறார்கள். எப்போதுமே அப்படி நடப்பதில்லை. நேற்று வேலை செய்த நிரல்கூறு இன்று வேலை செய்யாது, குழு உறுப்பினர்கள் உடல்நலக்குறைவு, எழுதும் நிரல் நீங்கள் எதிர்பார்ப்பதை செய்வதில்லை, வன்பொருள் சிக்கல்கள் ஆக ஒவ்வொரு நாளும் ஏதாவது பிரச்சினை வந்து கொண்டே இருக்கும்.

ஒரு குறுவோட்டத்தை நிர்வகிப்பதை ஒரு ஓடுதளத்தின் முடிவில் ஒரு போர் விமானத்தை தரையிறக்குவது போல எண்ணிப் பார்க்கலாம். நீங்கள் ஒரு போர் விமானத்தைத் தரையிறக்க ஒரே நேரத்தில் பல அளவீடுகளைக் கவனமாகப் பார்த்து சரி செய்ய வேண்டும். பறக்கும் வேகம், உயரம், ஓடுதளத்துடன் நேர்படுத்தல் மற்றும் குறுக்காக அடிக்கும் காற்று, இயந்திரத்தின் மீது அதிக சுமை போட்டு நின்று விடாமல் பார்த்துக்கொள்வது இவை எல்லாமே மிக முக்கியம். சிறிதும் முன்னே பின்னே இருக்க வழி இல்லை. ஆனால் இவை யாவற்றையும் ஒரே நேரத்தில் கூர்ந்து கவனித்து கட்டுப்படுத்துவதும் மிகக் கடினம். வானூர்தி இறங்கு பாதையில் கொஞ்சம் அதிக உயரத்தில் இருந்தாலும், நீங்கள் எளிதாக ஓடுதளத்தின் மத்தியில் இறங்கும் வாய்ப்பு உள்ளது. ஓடுதளம் ஈரமாக அல்லது பனியாக இருந்தால் அதன் முடிவில் போய் வானூர்தி நிற்க இயலாமல் சரிந்து விழவும் கூடும். இது எனக்கு இரண்டு முறை நடந்தது. நல்லவேளையாக இரண்டு

முறையும் விமானத்தில் இருந்த வால் கொக்கியை கீழே விட்டு ஓடுபாதையின் முடிவில் எப்போதும் இருக்கும் சங்கிலியை என்னால் பிடிக்க முடிந்தது. தவறியிருந்தால் விமானம் அதற்கப்பால் உள்ள தண்ணீரில் விழுந்திருக்கும் அல்லது மரங்களின் மேல் மோதியிருக்கும்.

இதே போல ஒரு குறுவோட்டத்தில் குழுவின் வேகம், நிரலின் தரம், இந்த வேகம் நெடுநாளைக்கு நிலைத்திருக்கக்கூடியதா, சரியான திசையில் செல்கிறோமா அதாவது கிடக்கும் பணிகளில் சரியானவற்றைத்தான் செய்கிறோமா, இவை யாவற்றையும் ஒரே நேரத்தில் கவனிக்க வேண்டும்."

இவையெல்லாம் ஒரே நேரத்தில் அணி உறுப்பினர்கள் கவனிக்க எளிதாக இருக்குமாறு எரிந்து குறையும் வரைபடத்தை (Burn down Chart) வடிவமைத்தார்.



இது மிக எளிய வளைகோடு, ஆனால் ஒரே நேரத்தில் மேற்கண்ட அனைத்து விஷயங்களைப் பற்றியும் கேள்விகள் எழுப்பும். குறுவோட்டத்தை முடிக்க அணி எடுக்கும் நடவடிக்கைகளின் திறனை தெளிவாகக் காட்டும். நாம் முந்தைய கட்டுரையில் கண்டது போல, பின்னர் வந்த ஆண்டுகளில் தகவெளிமை (Agile) செயல்முறைகளில் Scrum முன்னணி செயல்முறையாக இடம் பெற்றது.

*Scrum-க்கு தமிழாக்கம் 'மொய்திரள்' என்று விக்சனரி கூறுகிறது. எனக்கும் இது சரியாகவே படுகிறது. நீங்கள் என்ன நினைக்கிறீர்கள்? கீழே உள்ள கருத்துப்பெட்டியில் உங்கள் பரிந்துரைகளைப் பகிர்ந்து கொள்ளுங்கள்.

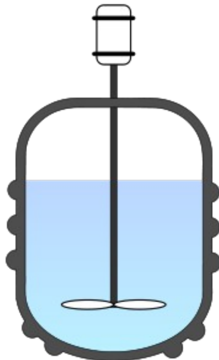
7. மென்பொருள் திட்டம் நிர்வகிக்க, மேம்பட்ட பாலிமர்கள் செய்யத் தெரிந்து கொள்ளுங்கள்!

எவ்வளவு முயன்றும் அருவி செயல்முறை எதிர்பார்த்த விளைவுகளைத் தராததால் தொண்ணூறுகளின் ஆரம்பத்தில் கென் ஷ்வாபர் *(Ken Schwaber)* மொய்திரள் *(Scrum)* முறையை ஜெஃப் சதர்லாண்ட் *(Jeff Sutherland)*-உடன் சேர்ந்து உருவாக்கி செயல்படுத்தினார். அதைப் பயன்படுத்தியதில் திட்டங்கள் வெற்றிக்குப் பின் வெற்றியாக முடிந்தன. மென்பொருள் திட்டங்களுக்கு என்ன அடிப்படை நெறிமுறைகளால் அருவி முறையை விட மொய்திரள் மிகவும் வெற்றிகரமாக வேலை செய்கிறது என்று தெரிந்து கொள்ள அவர் விழைந்தார்.

டூபாண்ட் நிறுவனத்தின் டெலாவேர் தொழில்நுட்ப மையத்தில் மேம்பட்ட செயல்முறை கட்டுப்பாடு வேலை செய்துகொண்டிருந்த அவரது நண்பர்களுடன் பேசினார். டூபாண்ட் நிறுவனம் ரேயான் போன்ற எளிய பாலிமர்கள் உற்பத்தியில் தேர்ச்சி பெற்றிருந்தது. அவர்களால் உலகத்தில் எங்கு வேண்டுமானாலும் அவற்றை உற்பத்தி செய்ய முடிந்தது. ஆனால் *GoreTex* போன்ற மேம்பட்ட பாலிமர்கள் உற்பத்தி மிகவும் சிக்கலானதாக இருந்ததால் பல இடங்களில் செய்ய முடியவில்லை. அவரது நண்பர்கள் இந்தப் பிரச்சினைக்கு தீர்வு காண ஆய்வு செய்து கொண்டிருந்தனர்.

எளிதில் கணிக்க முடியாத சிக்கலான செயல்முறைகளுக்கு அனுபவ செயல்முறை கட்டுப்பாடு *(empirical process control)* என்ற உத்தியே கையாளப்படுகிறது. மென்பொருள் உருவாக்குவதற்கு இதுவே தக்க செயல்முறை என்று பாலிமர் செயல்முறை நிபுணர்கள்

பரிந்துரைத்தனர். ஏனென்றால் தேவைகள் மாறிக்கொண்டே உள்ளன, தொழில்நுட்பங்களும் சிக்கலானவை மற்றும் வேலை செய்பவர்கள் படைப்பு சிந்தனை செய்ய வேண்டியுள்ளது. இந்த அணுகுமுறை எதிர்பாராதவைகளைச் சமாளிக்கத் தோதானது. முன்னேற்றத்தை அடிக்கடி ஆய்வு செய்து பின்னர் அடுத்த படிகளை அதற்குத்தக அமைப்பதன் மூலம் வெளியீட்டை மேம்படுத்தலாம் என்று கூறினர்.



கொள்கலன் என்பது ஒரு மூடிய இடம். இதற்குள், பிரச்சினையின் ஒட்டுமொத்த சிக்கலை பொருட்படுத்தாமல், வேலையை செய்து முடிக்க இயலும். மொய்திரள் செயல்முறையின் கொள்கலன் ஒரு குறுவோட்டம் (sprint). நாம் இந்தக் கொள்கலனில் வேலையைச் செய்யத் தேவையான அனைத்து திறமைகளும் உள்ள குழு உறுப்பினர்களை வைக்கிறோம். உயர்ந்த மதிப்புள்ள, தீர்வு காண வேண்டிய பிரச்சினைகளையும் நாம் இதே கொள்கலனில் வைக்கிறோம். உறுப்பினர்கள் பிரச்சினைக்குத் தீர்வு காண முழு முயற்சி செய்ய ஏதுவாக நாம் எந்த வெளித் தொந்தரவும் வராமல் கொள்கலனைப் பாதுகாக்கிறோம். பிரச்சினையில் வேலை செய்ய அனுமதிக்கும் நேரத்தை முன்னரே தீர்மானிப்பதன் மூலம் நாம் கொள்கலனைக் கட்டுப்படுத்துகிறோம். கொடுத்த நேரத்தில் எவ்வளவு வேலை முழுவதும் செய்து முடிக்க இயலும் என்று எதிர்பார்க்கிறார்களோ அந்த அளவு வேலையை

உறுப்பினர்கள் எடுத்துக் கொண்டால் போதும். கொடுத்த நேர முடிவில் நாம் கொள்கலனைத் திறந்து விளைவுகளை ஆய்வு செய்வோம். இதன் அடிப்படையில் நாம் கொள்கலனை அடுத்த குறுவோட்டத்துக்குத் தக மீட்டமைப்போம். இவ்வாறு அடிக்கடி மறுதிட்டமிடுதலால் நம்மால் திறனை அதிகப்படுத்த முடியும்.

இந்த முயற்சியின் மூலம் மொய்திரள் மற்றும் அதன் முக்கிய அங்கமான குறுவோட்டம் இரண்டுக்கும் கோட்பாடுகள் கொண்டு திடமான அடித்தளம் அமைத்தார்.

2010-ல் மொய்திரளுக்கு அறிவுத்தொகுப்பு இருந்தால் குறிப்புதவிக்கு எளிதாக இருக்கும் என்று ஜெஃப் சதர்லாண்ட்-உடன் சேர்ந்து மொய்திரள் கையேடு எழுதினார். மொய்திரள் ஒரு சட்டகம். ஆய்வு செய்யப்பட்ட அணுகுமுறைகள் உண்டு. பயன்படுத்தப்படும் விவரமான நடைமுறைகள், உத்திகள், செயல்முறைகள் கிடையாது. சட்டகங்கள் சக்தி வாய்ந்தவை. ஏனென்றால் அவற்றில்

போதுமான இணங்கு தன்மை இருப்பதால், நன்கு ஆய்வு செய்யப்பட்ட அணுகுமுறைகளைப் பயன்படுத்தி, மாறிவரும் நிலைமைகளுக்கு ஏற்ப அல்லது உங்கள் நிறுவனத்துக்கு தக அமைத்துக்கொள்ள முடியும். மொய்திரள் கையேடு மொத்தம் 16 பக்கங்கள்தான்! 2011-லும் பின்னர் 2013-லும் செயலாக்க அனுபவத்தின் அடிப்படையில் மொய்திரள் கையேட்டில் சில சிறிய மாற்றங்கள் செய்தனர்.

ஒரு அணியில், மொய்திரள் நடத்துனர் (Scrum Master) மற்றும் தயாரிப்பு உரிமையாளர் (Product Owner) தவிர, 3 முதல் 9 பேர் இருக்கலாம். மூன்றை விடக்குறைவாக இருந்தால் வேலையை முழுவதும் செய்து முடிக்கத் தேவையான திறமைகள் எல்லாம் இருக்குமா என்பது சந்தேகம். ஒன்பதை விட அதிகமாக இருந்தால் அணியை ஒருங்கிணைத்தல் கடினம். கருத்துக்களை பரிமாறுதல் மற்றும் தகவல்களை பகிர்தல் கடினமானதாக ஆகிறது.

மொய்திரளின் பயன்களைப் பற்றி கென் ஷ்வாபர் இவ்வாறு கூறுகிறார், "ஆர்வமுள்ள உருவாக்குநர்கள்* தங்கள் வாடிக்கையாளர்களுடன் நெருக்கமாக வேலை செய்து மிகவும் மதிப்புமிக்க, ஆக்கபூர்வமான மென்பொருட்களை உருவாக்க உகந்த வழி மொய்திரள். ஆனால் அருவி செயல்முறைக்கான கலாச்சாரத்தை பின்பற்றிய நிறுவனங்கள் மாறுவது மிகக் கடினமான பாதையாக உள்ளது. அக்கலாச்சாரத்தில் கட்டளையும் கட்டுப்பாடும் மற்றும் இப்படித்தான் செய்ய வேண்டும் என்ற விதிமுறைகளும் அதிகம். எனினும் இந்தப் பாதையை அவர்கள் கடந்து வர முடிந்தால் வெகுமதிகளோ மிகப் பெரியவை. இந்த மாற்றத்தை மெனக்கெட்டு செய்யும் நிறுவனங்கள் சந்தைப் போட்டியில் வெல்வர், மற்றும் நிரலாளர்கள் அனைவரும் அங்கு வேலை செய்ய விரும்புவர்."

இதனால்தான் மொய்திரளை புரிந்து கொள்வது எளிது ஆனால் செயலாக்கத்தில் தேர்ச்சி பெறுவது கடினம் என்று கையேடு சொல்கிறது.

* உருவாக்குநர்கள் என்ற பொதுவான சொல்லில் நிரலாளர்கள், சோதனையாளர்கள், வடிவமைப்பாளர்கள் மற்றும் மென்பொருள் திட்டங்களில் வேலை செய்யும் யாவரையும் சேர்த்தே குறிப்பிடுகிறோம்.

8. உற்பத்தித் திறனை மேம்படுத்த குமிழிகள் அனைத்தையும் 10-க்குத் திருப்புங்கள்!

கென்ட் பெக் (Kent Beck) 1996-ல் க்ரைஸ்லர் நிறுவனத்தின் சம்பள செயலியை மாற்றி எழுதும் மென்பொருள் திட்டத்தின் தலைவராக ஆனார். அந்தத் திட்டத்தில் அதீத நிரலாக்கத்தை (Extreme Programming or XP) ஆரம்பித்து செயற்படுத்தினார். 1999-ல் அவர் Extreme Programming Explained புத்தகத்தை வெளியிட்ட போது அதில் அவர் விவரித்த சில நடைமுறைகள் மென்பொருள் உலகை அதிர வைத்தன.

அனைத்து மென்பொருளையும் இரண்டு நிரலாளர்கள் இணைந்து உருவாக்குதல் – அதாவது ஒரு திரை இரண்டு நிரலாளர்கள் *(pair programming)*. ஒருவர் எழுதிய நிரலை குழுவில் மற்றொரு சக நிரலாளர் மறுஆய்வு செய்வது வழுக்களைக் குறைத்து நிரலின் தரத்தை மேம்படுத்தும் என்பதற்கு நிறைய ஆதாரங்கள் உள்ளன. ஆனால் இரண்டு நிரலாளர்கள் இணைந்து உருவாக்குதல் மிகவும் புரட்சிகரமான நடவடிக்கை.

முதலில் ஓரலகு சோதிப்புகள் *(unit tests)* எழுதிவிட்டு பின்னர்தான் நிரல் எழுதவே ஆரம்பித்தல். நிரலாளர்கள் நிரல் எழுதியவுடன் கைமுறையில் சோதனை செய்வதே வழக்கமாக இருந்தது. இந்த சோதனையை தானியங்கியாக செய்தல் நல்ல மேம்பாடுதான். ஆனால் முதலில் சோதனை செய்துவிட்டு பின்னர்தான் நிரல் எழுதவே ஆரம்பித்தல் என்பது நூதனமானது.

மறுசீரமைப்பு *(refactoring)* செய்து நிரலை தொடர்ந்து மேம்படுத்தல். உற்பத்தி மற்றும் கட்டுமான தொழில்துறைகளில் "முதல் முறையே சரியாகச் செய்வோம்" என்ற முழக்கம் நன்கு நிலைபெற்றது. நாம் முன்னால் பார்த்தபடி இத் தொழில்துறைகளில் வேலையை ஒரு முறை செய்த பிறகு அதை மாற்றியமைப்பது மிகக் கடினம். ஆனால் மென்பொருளில் நிரலை மாற்றி அமைப்பது அவ்வளவு கடினம் அல்ல. ஆனால் மாற்றி அமைத்த பின் சோதனை செய்வதுதான் வேலை கொள்ளும், மேலும் இடர்பாடானது. நிரலாளர்களைக் கேட்டுப் பாருங்கள். ஐந்தே நிமிடத்தில் மாற்றித் தருகிறேன் என்றுதான் சொல்வார்கள். ஆனால் சோதனையாளர்களோ வேறு எங்கு ஒழுங்காய் வேலை செய்து கொண்டிருக்கும் நிரல்கூறு உடையுமோ என்று கவலை கொள்வார்கள்.

சில சமயங்களில் ஒரு சிறிய வழுவை நீக்கப் போய் அது பெரிய பிரச்சினைகளில் கொண்டு விடுவதை நாம் பார்த்திருக்கிறோம், நாம்

அன்றாடம் பயன்படுத்தும் செயலிகளில் நடைமுறையிலும் அனுபவித்திருக்கிறோம். எடுத்துக்காட்டாக 2005 ஆம் ஆண்டில், டிரெண்ட் மைக்ரோ (Trend Micro) தரமற்ற பேட்ச் (patch) ஒன்றை வெளியிட்டது. இதனால் செயல்திறன் கடுமையாகக் குறைந்தது. இந்தப் பிரச்சினையால் அவதிப்பட்ட வாடிக்கையாளர்களிடம் இருந்து சுமார் 4 லட்சம் புகார்கள் வந்து குவிந்தன. இறுதியில் வாடிக்கையாளர்களுக்கு இழப்பீடு மட்டும் $ 8 மில்லியனுக்கு மேல் செலவிட நேர்ந்தது.

XP மாற்றி அமைப்பதை ஊக்குவித்தது மட்டும் அல்ல, அதை அவசியமாக்கியது. ஏனென்றால் ஒரு குறிப்பிட்ட தேவைக்கு நீங்கள் குறைந்தபட்ச நிரல் எழுதி ஓரலகு சோதிப்பில் சரியானால் போதும். அடுத்து நிரலை மறுசீரமைப்பு செய்யும்போது அதை மேம்படுத்தலாம். XP-ல் சோதனை செய்வது கடினம் அல்ல. ஏனென்றால் இப்போது உங்களிடம் ஓரலகு சோதிப்பின் முழுத்

தொகுப்பும் உள்ளது. அதுவும் நீங்கள் பொத்தானைத் தட்டி விட்டால் தானாகவே ஓடும் வகையில் நிறுவி வைத்திருப்பீர்கள். ஆக, நிரல் மறுசீரமைப்பு மென்பொருளில் தொடர்ந்து முன்னேற்றம் செய்ய வழி செய்தது. ஓரலகு சோதிப்பு மறுசீரமைப்பினால் புதிய பிரச்சினைகள் வராது என்ற நம்பிக்கையைக் கொடுத்தது.

ஒவ்வொரு நாளும் பல முறை தொடர் ஒருங்கிணைப்பு செய்தல் (continuous integration). சில அருவி செயல்முறை திட்டங்களில் ஒருங்கிணைத்தல் செய்ய கடைசி கட்டங்களில்தான் நேரம் ஒதுக்கப்பட்டிருக்கும். அப்படி செய்யும்போது ஒருங்கிணைத்தலிலும் அதன்பின் சோதனையிலும், முக்கியமாக செயல்திறன் சோதனையிலும், பல பிரச்சினைகள் வெளிப்படலாம். முன்னெச்சரிக்கையான சிலர் இடையிடையே ஒருங்கிணைத்தல் செய்வது உண்டு. விடாமுயற்சி உள்ள சிலர் ஒவ்வொரு வாரமும்

வாரக்கடைசியில் ஒருங்கிணைத்தல் செய்வர். ஆனால் ஒவ்வொரு முறையும் நிரலாளர் தொகுபதிவகத்தில் நிரல் சேர்க்கும்போது தானியங்கியாக ஒருங்கிணைத்தலும், சோதனை செய்தலும் அதுவரை கேட்டறியாத விஷயம்.

அவர் அதீத நிரலாக்கத்தை இப்படித்தான் விளக்கினார். மென்பொருள் உருவாக்கத் தேவையான ஒவ்வொரு செயல்முறையையும் கட்டுப்பாட்டகத்தில் ஒரு குமிழி என்று உருவகப்படுத்திக் கொண்டார். தான் எந்த செயல்முறைகள் இன்றியமையாதவை என்று நினைத்தாரோ அவற்றை 1 முதல் 10 வரை உள்ள டயல்களில் அனைத்துக் குமிழிகளையும் 10-ல் திருப்பி வைத்தால் என்ன ஆகிறதென்று பார்க்கலாம் என்றார்.



இவை எல்லாவற்றையும் செயல்படுத்திய பின் இந்த செயல்முறைகளின் முழுத் தொகுப்பு நிலையானதாக, கணிக்கக் கூடியதாக, மற்றும் இணங்கு தன்மையுடன் இருந்ததைப் பார்த்து அவருக்கே கொஞ்சம் ஆச்சரியமாக இருந்தது என்றார்.

ஆக XP பொது அறிவுக் கொள்கைகளை அதீத மட்டங்களுக்கு எடுத்துச் செல்கிறது. மேலும் எடுத்துக்காட்டாக குறுகிய மறுசெய்கை

*(iteration)** நல்லது என்றால் அதைக் கூடியவரை *XP* குறுக்குகிறது. இதற்காக குறைந்தபட்ச ஆனால் மிகவும் வணிக அர்த்தமுள்ள, சாத்தியமான வெளியீட்டை தேர்வு செய்ய வாடிக்கையாளரைக் கேட்டுக்கொள்கிறது. அருவி செயல்முறையில் பெரும்பாலும் திரைப்பட வெளியீடு போல தடல்புடலாக முழு மென்பொருளையும் ஒரே நேரத்தில் வெளியீடு செய்வதுடன் ஒப்பிட்டுப் பாருங்கள்.

XP இரண்டு விதமான வாக்குறுதிகளை அளிக்கிறது என்று சொல்கிறார் கென்ட் பெக், “நிரலாளர்களுக்கு ஒவ்வொரு நாளும் முக்கியமான விஷயங்களில் வேலை செய்ய முடியும் என்று *XP* வாக்களிக்கிறது. வாடிக்கையாளர்கள் மற்றும் மேலாளர்களுக்கு *XP* ஒவ்வொரு வாரமும் அதிகபட்ச மதிப்புடைய மென்பொருள் பெற முடியும் என்று வாக்களிக்கிறது. மேலும் அவர்கள் அநியாய செலவுகள் செய்யாமல் திட்டத்தின் மத்தியில் திசையை மாற்ற முடியும். சுருக்கமாகச்

சொல்லப்போனால் பணித்திட்டத்தின் இடரைக் குறைக்கவும், வணிக மாற்றங்களுக்குத் தக மாற்றவும், உற்பத்தித் திறனை மேம்படுத்தவும், நிரலாக்க அணி ஆர்வத்துடன் ரசித்து வேலை செய்யவும், இவை யாவையும் ஒரே நேரத்தில் முடியும் என்று XP வாக்களிக்கிறது".

**Scrum-ல் 'sprint' என்று சொல்வதை XP-ல் 'iteration' என்று சொல்கிறோம். இதுவும் அதே குறுவோட்டம்தான்.*

9. லேத் பட்டறையில் வேலையை மாற்றி மாற்றி ஏற்றி இறக்குவது போல!

நான் தகவல் தொழில்நுட்பத்துக்கு மாறுவதற்கு முன் ஒரு இயந்திர பொறியாளராக என் தொழில் வாழ்க்கையைத் தொடங்கினேன். நடுத்தர அளவிலான தொழிற்சாலையில் வேலை திட்டமிடல் எப்படி சிறப்பாகச் செய்யலாம் என்று பார்த்துக் கொண்டிருந்தேன். லேத் பட்டறை இயக்குபவரிடம் பேசிக்கொண்டிருந்த போது சொன்னார், ‘‘காலைல கிருஷ்ணன் சார் சொல்ற வேலையை ஏத்துவோம் சார். அதைக் கஷ்டப்பட்டு அலைன் பண்ணி முடிச்சு டூல் செட் பண்ணி கடைசல ஆரம்பிக்கலாம்னு இருப்போம். அப்பதான் சக்திவேல் சார் வந்து

அதை இறக்கிட்டு வேற வேலைய ஏத்தச் சொல்லுவாரு. அப்புறம் மூர்த்தி சார் ரவுண்டு வருவாரு... ஆக கடைசல் பிடிக்கறது கம்மிதான் சார். ஏத்தி இறக்குற வேலைதான் ஜாஸ்தி.''

சில மேலாளர்கள் தங்களிடம் வந்த வேலைகளையெல்லாம் கீழே வேலை செய்பவர்களிடம் அனுப்பிக்கொண்டே இருப்பார்கள். முன்காலத்திலாவது பார்த்தால் கோப்புகள் மேசை மேல் அடுக்காகத் தெரியும். இப்பொழுது அதுவும் தெரிவதில்லை – மின் அஞ்சலிலோ அல்லது மின் ஆவணங்களிலோ கண்ணுக்குத் தெரியாமல் மறைந்து உள்ளன. எல்லா வேலையும் அவசரம், உடனே வேண்டும், நேற்றே வேண்டும். ஒரே ஒரு தலைவர் இருக்கும்போதே இப்படி என்றால் சிலருக்குப் பல தலைவர்கள் உண்டு. ஒவ்வொருவரும் தன் வேலையை முதலில் செய்து தர வற்புறுத்துவர்.

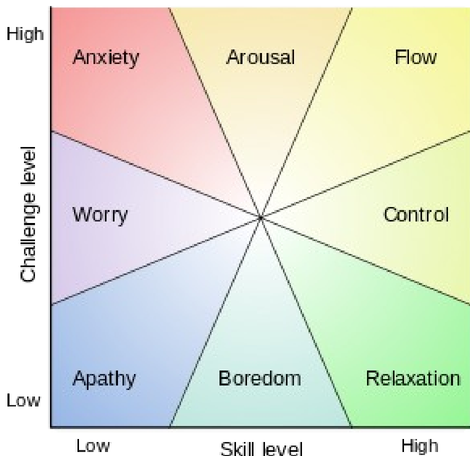
இது போதாதென்று வேலை செய்பவர்கள் சிலரும் தாங்கள் எவ்வாறு பல்பணியாக்கத்தில் அசகாய சூரன் என்றும் அஷ்டாவதானி என்றும் சொல்லி மார் தட்டுவர்.

ஆனால் ஆராய்ச்சிகளின்படி பல்பணியாக்கம் உற்பத்தித் திறனை மிகக் குறைக்கிறது. ஏனென்றால் நம்முடைய மூளை பல்பணியாக்கத்திற்கு தோதாக செய்யப்படவில்லை. நாம் பல்பணியாக்கம் செய்வதாக நினைக்கும்போது உண்மையில் ஒரு பணியிலிருந்து மற்றொரு பணிக்கு மிக வேகமாக முன்னும் பின்னும் தாவுகிறோம். நிரலாளர்கள் மட்டுமல்ல, அறிவு சார்ந்த வேலை செய்பவர் எவருமே எடுத்த வேலையை முழுவதும் முடித்துவிட்டு அடுத்த வேலைக்கு செல்வதே உற்பத்தித் திறனுக்கு சிறந்தது, அவர்களுடைய மன அழுத்தத்துக்கும் உகந்தது. ஆகவே தலை போகிற அவசரம் என்றாலொழிய செய்து கொண்டிருக்கும் வேலையை முடித்து

விட்டு அடுத்த வேலைக்குச் செல்வதுதான் உசிதம்.
நிரலாக்கம் செய்வதற்கு நிரலாளர்கள் ஒரே நேரத்தில் நிறைய விஷயங்களை குறுகிய கால நினைவில் வைத்துக் கொள்ள வேண்டும். எவ்வளவு விஷயங்கள் அவர்களால் நினைவில் வைக்க முடிகிறதோ அவ்வளவுக்கு அவர்கள் நிரலாக்க உற்பத்தித் திறன் அதிகம். முழு வேகத்தில் வேலை செய்யும் நிரலாளர்கள் எண்ணிலடங்கா விஷயங்களை தங்கள் நினைவில் வைத்து இருப்பார்கள் – மாறிகளின் பெயர்களில் இருந்து, தரவுக் கட்டமைப்புகள், முக்கியமான *API*-கள், அவர்கள் எழுதிய மற்றும் நிறைய அழைக்கும் பயனுடை செயல்பாடுகளின் பெயர்கள், இத்யாதி... இத்யாதி.
இதைத்தான் மன ஒருமை அல்லது முழுமையான ஒருமுக சிந்தனை என்று கூறுகிறோம். இதை ஆங்கிலத்தில் *"flow"* அல்லது *"in the zone"* என்று கூறுகிறார்கள். இந்த நிலையில் வேலை செய்பவர்கள் தங்களைச்

சுற்றி என்ன நடக்கிறதென்றே தெரியாமல் முழுமையாகத் தங்கள் பணியில் கவனம் செலுத்துவார்கள். நேரம் காலமே தெரியாமல் கடினமான வேலைகளையும் செய்து முடிப்பார்கள். எழுத்தாளர்கள், விஞ்ஞானிகள், மற்றும் விளையாட்டு வீரர்கள் கூட மன ஒருமையைப்பற்றி சொல்வார்கள்.

அருவி செயல்முறையில் தேவைப் பட்டியலில் முன்னுரிமை ஒன்று, இரண்டு, மூன்று என்று குறிப்பிடுவது வழக்கம். ஆனால் பலர் பெரும்பாலான தேவைகளை முன்னுரிமை ஒன்று என்றே குறிப்பிடுவர்.



மேற்கண்ட காரணங்களால்தான் மொய்திரள் விதிமுறை கிடக்கும் வேலைகளை வரிசைப்படுத்தச் சொல்கிறது. அதுவும் தயாரிப்பு உரிமையாளருக்கு மட்டும்தான் வரிசைப் படுத்தும் அதிகாரம் உண்டு. வேறு யாராக இருந்தாலும் முன்னுரிமை தேவைப்பட்டால் தயாரிப்பு உரிமையாளரிடம்தான் பேச

வேண்டும். குறுவோட்டம் ஆரம்பிக்கும் வரை தயாரிப்பு உரிமையாளர் தலைகீழ் மாற்றங்கள் கூடச் செய்யலாம். ஆனால் வேலையை ஆரம்பித்தபின், அதாவது குறுவோட்டம் ஆரம்பித்தபின், தயாரிப்பு உரிமையாளர் கூட எந்த மாற்றமும் செய்ய முடியாது. இதுதான் மொய்திரளின் தனிச்சிறப்பு.

கென் ஷ்வாபர் குறுவோட்டத்தை ஒரு கொள்கலன் போன்று உருவகப்படுத்தியதை நினைவில் கொள்ளுங்கள். குழு உறுப்பினர்கள் பிரச்சினைக்குத் தீர்வு காண முழு முயற்சி செய்ய ஏதுவாக எந்த வெளித் தொந்தரவும் வராமல் மொய்திரள் நடத்துநர் கொள்கலனைப் பாதுகாக்க வேண்டும்.

இதற்கு வழக்கமான விதிவிலக்கு ஒன்றே ஒன்று உண்டு. ஓடிக்கொண்டிருக்கும் தயாரிப்பில் வழு வெளிப்பட்டு பயனர்கள் வேலை செய்வது தடைப்பட்டால் அதற்கு முன்னுரிமை கொடுத்தே ஆக வேண்டும்.
குறுவோட்டத்துக்குத் திட்டமிடும்போது சிலர்

முன்யோசனையாக சுமார் 5 முதல் 10% நேரத்தை தயாரிப்பில் வழு வெளிப்பட்டால் சரி செய்வதற்கு ஒதுக்குவர். சிலர் திட்டமிட்ட வேலையில் ஒன்றை வெளியில் எடுத்துவிட்டு இம்மாதிரி நெருக்கடி வழுச் சீட்டை குறுவோட்டத்தில் சேர்ப்பர். ஆனால் அத்தியாவசியமான வழுவாக இருந்தால் மட்டுமே இந்த மாற்றம் செய்யலாம். வேறு ஏதாவது மாற்றம் செய்தே ஆக வேண்டுமென்றால் என்ன செய்வது? அசாதாரணமான சூழ்நிலைகளில், வேறு வழியே இல்லையென்றால், தயாரிப்பு உரிமையாளர் முழுக் குறுவோட்டத்தையும் ரத்து செய்து விட்டு அடுத்தக் குறுவோட்டத்தைப் புதிதாகத் தொடங்கலாம். ஆனால் இது குழுவுக்கு அதிர்ச்சிகரமான நடவடிக்கை. மிகவும் கவனமாகப் பயன்படுத்த வேண்டும்.

10. ஒருக்கால் தேவைப்படலாம் என்று எவ்வளவு தேவையற்ற வேலைகள் செய்கிறோம்!

“செய்யாத வேலையை முடிந்த அளவுக்கு அதிகப்படுத்தும் கலை, இன்றியமையாதது.” மென்பொருள் உருவாக்குவதற்கான கொள்கை விளக்க அறிக்கையுடன் (Agile Manifesto) வெளியிட்ட 12 கோட்பாடுகளில் ஒன்று இது.

இது விநோதமாக இல்லை? இவர்கள் சோம்பேறித்தனத்தை ஊக்குவிக்கிறார்கள் போல் அல்லவா தோன்றுகிறது! ஆனால் இதன் அர்த்தம் அதுவல்ல. பின்னால் தேவைப்படலாம்

என்று வேலையை இழுத்துப் போட்டுகொண்டு செய்யாதீர்கள். அது தேவைப்படாமல் போகலாம். எடுத்துக்காட்டாக *You Aren't Gonna Need It (Yagni)* "இது உங்களுக்குத் தேவைப்படாது" என்பது அதீத நிரலாக்கத்தில் ஒரு கொள்கை.

கென்ட் பெக் *(Kent Beck) – உடன் 1996-ல்* மென்பொருள் திட்டத்தில் வேலை செய்து அதீத நிரலாக்கம் உருவாக்க உதவியவர் ரான் ஜெஃப்ரீஸ் *(Ron Jeffries).* அவர் எடுத்துக்காட்டுகளுடன் மேற்கண்ட கொள்கையை விளக்குகிறார், "பெரும்பாலும் நீங்கள் நிரல் எழுதிக் கொண்டிருக்கும் போது உங்கள் மனதுக்குள் நமக்கு இன்னது பின்னால் தேவைப்படும் என்று தோன்றும். இந்த உந்துதலுக்கு எப்பொழுதும் செவிமடுக்காதீர்கள். அதாவது நிரலாளர்கள் எந்த செயல்பாட்டையும் அது அவசியம் தேவைப்படும் போதுதான் செய்ய வேண்டும். அது பின்னால் தேவைப்படலாம் என்று

எதிர்பார்த்து முன்னேற்பாடாக செய்து வைத்துக்கொள்ளக் கூடாது.'' இது இன்னும் விநோதமாக இல்லை? வேலையைத் தள்ளிப்போடும் குணத்தை ஊக்குவிக்கிறார்கள் போல் அல்லவா தோன்றுகிறது! வந்தபின் கவலைப்படுவதை விட வருமுன் காப்பது மேல் அல்லவா?

கென்ட் பெக் (Kent Beck) – உடன் சேர்ந்து 1999-ல் Refactoring: Improving the Design of Existing Code என்ற புத்தகத்தை எழுதிய மார்ட்டின் ஃபௌலர் (Martin Fowler) ஏனென்று மேலும் விளக்குகிறார், "XP-ன் மற்ற பல கொள்கைகள் போலவே, அது 90 களின் இறுதியில் மென்பொருள் பொறியியலில் பரவலாக நம்பப்பட்ட கொள்கைகளுக்கு மிகவும் மாறாகத்தான் இருந்தது. கவனமாக முன் திட்டமிடல் மென்பொருள் உருவாக்குவதற்கு முக்கியம் என்று மிக அதிகமான உந்துதல் அந்த நேரத்தில் இருந்தது. பெரிய அம்சங்களில்தான் Yagni நன்றாகத் தெரியும், ஆனால் நீங்கள் சிறிய விஷயங்களிலும் இதை அடிக்கடி பார்க்க

முடியும். சில சிறிய *Yagni* முடிவுகள் திட்ட மேலாண்மையின் கவனத்திற்கே வராது. ஒரு நிரலாளர் விரைவில் தேவைப்படலாம் என்று நினைத்து ஒரு சிறிய விஷயத்தில் எளிதாக ஒரு மணி நேரம் செலவிடக்கூடும். இம்மாதிரி சிறிய *Yagni* முடிவுகள் நிறையச் செய்தால் நிரலில் சிக்கலை கணிசமாகக் குறைப்பது மட்டுமல்லாமல் அவசரமாகத் தேவையான அம்சங்களை சீக்கிரம் முடித்து வெளியிடவும் இயலும்.

ஆனால் தேவைப்பட்டபோது நிரலை மாற்றுவது எளிதாக இருந்தால்தான் *Yagni* ஒரு சாத்தியமான உத்தி ஆகும். இது சாத்தியமாவதற்கு, நீங்கள் ஓரலகு சோதனைகள் எழுதி, சோதனைத் தொகுப்பு ஓட்டுவதற்குத் தயாராக உங்கள் கையில் இருக்க வேண்டும். மற்றும் தொடர் ஒருங்கிணைப்பு செய்தலும் அவசியம்”.

இது இப்படி இருக்க, அருவி செயல்முறையில் திட்டம் ஆரம்பித்த பின் ஒரு தேவையை மாற்றவோ அல்லது சேர்க்கவோ நீங்கள் முயற்சி செய்ததுண்டா? ஒருக்கால் நீங்கள் முயற்சி செய்திருந்தால் அது எவ்வளவு தலைவலியான வேலை என்று உங்களுக்குத் தெரிந்திருக்க வாய்ப்பு உண்டு. இந்தக் காரணத்தினால் பயனர்கள் தங்கள் தேவைகளைப் பட்டியலிட ஒரே ஒரு வாய்ப்பு மட்டும்தான் கிடைக்கும் என்று நினைக்கிறார்கள். ஆரம்பத்திலேயே குறைந்தபட்ச வெளியீட்டுக்கு மேல் மிக அதிகமாகத் தேவைகளைச் சேர்க்கிறார்கள். இதனால் என்ன பிரச்சினைகள் ஏற்படுகின்றன என்று பார்ப்போம்.

- நான்கு நிறுவனங்களில் அவர்களே செய்து பயன்படுத்தும் திட்டங்களில் எடுத்த கணக்கெடுப்பில் 64 சதவீதம் அம்சங்கள் "எப்போதாவது பயன்படுத்தப்படுகிறது அல்லது பயன்படுத்துவதே இல்லை" என்று தெரிய வந்தது.

- *டூபாண்ட் நிறுவனத்தில் செய்த ஆய்வில் 25% அம்சங்கள் மட்டும்தான் உண்மையிலேயே தேவை என்று தெரிய வந்தது.*
- *இங்கிலாந்தில் ஒரு பெரிய நிறுவனத்தில் 10 ஆண்டுகளாக பயன்படுத்திய நிரலியில் ஒரு ஆய்வு செய்தனர். ஆய்வின்படி சுமார் 10% -க்கு மேலான அம்சங்கள் ஒருபோதும் பயன்படுத்தவில்லை என்று தெரிய வந்தது. மற்றொரு 14% அம்சங்கள் 10 ஆண்டுகளில் ஒரே ஒரு தடவை மட்டுமே பயன்படுத்தப்பட்டுள்ளன. அடுத்த 25% அம்சங்கள் ஒரு டஜனுக்கும் குறைவான முறைகளே பயன்படுத்தப்பட்டுள்ளன. மொத்தத்தில், அமைப்பில் கிட்டத்தட்ட பாதி அம்சங்கள் பயன்படுத்தவே இல்லை அல்லது ஆடிக்கு ஒரு முறை, அமாவாசைக்கு ஒரு முறைதான் பயன்படுத்தப்பட்டன.*

மேலும் பயனர்கள் ஒரு அம்சம் தேவை என்று பட்டியல் தயார் செய்யும்போது அவர்களுக்கு அந்த அம்சத்தின் விலை என்ன என்றே தெரியாது. விற்பனையாளர்களும் ஒவ்வொரு

அம்சத்துக்கும் தனித்தனியாக விலை இன்னது என்று பிரித்துச் சொல்வதில்லை. தனித்தனியாக விலை தெரியாமல் பட்டியலுக்கு மொத்த விலை மட்டும் கேட்டு யாராவது சாமான் வாங்குவது உண்டா?

ஆக, பின்னால் கேட்பது பிரச்சினை என்றும் மேலும் விலை தெரியாமலும் பயனர்கள் சில அம்சங்களைக் கேட்க, ஒருக்கால் தேவைப்படலாம் என உருவாக்குநர்கள் சில சட்டகங்கள் கட்ட, அருவி செயல்முறையில் கண்ணுக்குத் தெரியாமல் தேவையற்ற வேலைகள் பல செய்கிறோம்.



YAGNI

It may look like overkill, but I'm sure we'll need it eventually.

மொய்திரள் செயல்முறையில் தயாரிப்பு உரிமையாளர் வேலைகளை வரிசைப்படுத்தும் போது அதன் மதிப்பீட்டைப் பார்த்து, செய்யும் செலவுக்கு அதிக பயன் தரும் வேலைகளுக்கு முன்னுரிமை தர முடிகிறது. தேவையற்ற அம்சங்களை செய்தால் குறுவோட்டம் முடிந்தவுடன் பங்குதாரர்கள் பார்த்து கேள்வி எழுப்புவார்கள். மேலும் நீங்கள் XP பொறியியல் நடைமுறைகளான ஓரலகு சோதிப்புகள் மற்றும் தொடர் ஒருங்கிணைப்பு செய்தல் ஆகியவற்றை பின்பற்றினால் உங்களால் தேவைப்பட்டபோது நிரலை எளிதில் மாற்றியமைக்க இயலும். ஒருக்கால் தேவைப்படலாம் என முன்னாலேயே செய்து வைத்துக்கொள்ள வேண்டிய அவசியமில்லை.

11. அருவி செயல்முறையிலிருந்து மொய்திரளுக்கு (Scrum) நிலைமாற்றம் செய்வது எப்படி?

“இதெல்லாம் சரிதான். அருவி செயல்முறையைக் கைவிட நாங்கள் (ஒரு மாதிரி) தயார்! கான்ட் வரைபடம் இல்லாமல் திட்டத்தை எப்படியாவது ஓட்ட முயற்சிக்கிறோம். இப்போது நாங்கள் தகவெளிமை (Agile) / மொய்திரள் (Scrum) – க்கு எப்படி நிலைமாற்றம் செய்வது என்று ஒரு சாத்தியமான வழியைச் சொல்லுங்கள்.” என்று நீங்கள் கேட்டது காதில் விழுந்தது!

Agile Alliance உலகம் முழுவதும் 5000-ம் பேரிடம் கருத்துக் கணிப்பு செய்து வெளியிட்ட அறிக்கையில் மொய்திரளின் நன்மைகளை அளவிடுவதுதான் மிகப்பெரிய சவாலாக இருக்கிறது என்று சொல்கிறார்கள். ஆகவே உங்கள் வணிக பிரச்சினைகள் எவை என்று முதலில் தெளிவாக்குங்கள். இந்த பிரச்சினைகளை சமாளிப்பதற்கு எந்த வணிகக் குறிக்கோள்கள் முக்கியம் என்று பாருங்கள்.

- கொடுத்த கெடுவில் திட்டத்தை முடிப்பது.
- கொடுத்த செலவுத் திட்டத்தில் முடித்துக் கொடுப்பது.
- வாடிக்கையாளர் கேட்கும் மாற்றங்களைக் கூடிய சீக்கிரம் செய்து தருவது.
- மென்பொருளின் தரத்தை உயர்த்துவது – அதாவது வெளியீட்டில் வரும் வழுக்களின் எண்ணிக்கையையும் தீவிரத்தையும் குறைப்பது.

- பணியாளர்களைத் தக்கவைப்பது.

- அணியின் உற்பத்தித்திறனை உயர்த்துவது.

இவற்றில் உங்களுக்கு இன்றியமையாததும் மற்றும் அளவிடக்கூடியதுமாக ஒன்றைத் தேர்ந்தெடுத்து கூடியவரை துல்லியமாக அளவிடுங்கள். இது நிலைமாற்றத்தை செயல்படுத்தும் முன் எடுக்கும் அளவு. ஆகவே இதை தளநிலையாக வைத்து நிலைமாற்றத்தின் பின் ஒப்பீடு செய்வதற்கு உகந்ததாக இருக்கும்.

மிகவும் சிறியதாகவும் இல்லாமல் மிகவும் பெரியதாகவும் இல்லாமல் ஒரு திட்டத்தைத் தேர்வு செய்யுங்கள். மிகவும் சிறியதாக இருந்தால் நிரூபிக்க உகந்ததல்ல. மிகவும் பெரிய திட்டத்தில் சோதனை வேண்டாம். வாடிக்கையாளரின் ஒத்துழைப்பு மிக அவசியம். அவர்கள் குறைந்தபட்ச சாத்தியமான மென்பொருளை வெளியிடத் தயாராக இருக்க வேண்டும். மற்றும் ஒவ்வொரு

குறுவோட்டத்தின் முடிவில் தயாரான மென்பொருளின் செயற்காட்சி பார்த்து பின்னூட்டம் தர முடிவு எடுக்கக் கூடிய மூத்த நிர்வாகிகள் முன் வர வேண்டும்.

சில முக்கிய உறுப்பினர்களை மொய்திரள் பயிற்சிக்கு அனுப்புங்கள். மொய்திரள் நடத்துநர் மற்றும் தயாரிப்பு உரிமையாளர் பயிற்சி அவசியம் தேவை. அடுத்து அவர்களை மற்றக் குழு உறுப்பினர்களுக்குப் பயிற்சி கொடுக்கச் சொல்லுங்கள்.

மொய்திரளுக்கு எந்தக் கருவிகளைப் பயன்படுத்துகிறீர்கள் என்பது அவ்வளவு முக்கியமல்ல. மொய்திரள் ஆர்வலர்கள் வெறும் வெண்பலகையும் குறிப்பு ஒட்டுத்தாளும் வைத்து ஆரம்பிப்பதுதான் சிறந்தது என்று கூறுவர். அவ்வாறு செய்வது நல்லதுதான், அணியின் கவனமெல்லாம் மொய்திரளில் இருக்கும். ஆனால் இதில் முக்கியப் பிரச்சினை அறிக்கைகளும் புள்ளிவிபரங்களும் தயார் செய்வது கடினம். மேலும் அணி உறுப்பினர்கள்

யாவரும் ஒரே இடத்தில் இல்லையென்றாலும் இணைய மென்பொருள் தேவை. ஆகவே ஏதாவதொரு இலவச கட்டற்ற மென்பொருள் வைத்து ஆரம்பிக்கலாம். பின்னர் அவசியம் தேவைப்பட்டால் இந்த அனுபவத்தை வைத்து வேறு மென்பொருள் தேர்ந்தெடுக்கலாம்.



முதலில் திட்டத்தின் இடரைக் குறைப்பதும், பணியாளர்கள் ஆர்வமுடன் வேலை செய்ய ஏதுவாக்குவதும் முக்கியம். இதற்கு அடிப்படைகளில் கவனம் செலுத்துங்கள்.

- திட்டத்தின் இடரைக் குறைக்க இயன்றவரை அடிக்கடி வெளியீடு செய்து பயனர்கள் மற்றும் பங்குதாரர்களிடம் பின்னூட்டம் பெறுங்கள்.

- தயாரிப்பு உரிமையாளர் மட்டும்தான் கிடக்கும் பணிகளை வரிசைப் படுத்தலாம்.

- குறுவோட்டத்துக்கு திட்டமிடுவதற்கு முன் தயாரிப்பு உரிமையாளர் உருவாக்குநர் குழுவுடன் ஒத்துழைத்து கிடக்கும் பணிகளை செம்மைப் படுத்த வேண்டும். ஏற்பு நெறிமுறைகள் தெளிவாக இருக்க வேண்டும்.

- குறுவோட்டத்தில் உருவாக்குநர்கள் முழுமையாக முடிக்கக்கூடிய அளவு மட்டுமே வேலையை எடுத்துக் கொள்ளுங்கள்.

- குறுவோட்டம் ஆரம்பித்த பின் எந்த மாற்றமும் செய்யாமல் பார்த்துக் கொள்வது மொய்திரள் நடத்துநரின் பொறுப்பு.

- எந்த வேலையை யார் செய்வது என்று உருவாக்குநர்கள் சுயமாகப் பங்கிட்டுக்கொள்ள வேண்டும். வெளியிலிருந்து எவரும், மொய்திரள் நடத்துநர் கூட, தலையிடக்கூடாது.

- ஒருவர் எழுதிய நிரலை குழுவில் மற்றொரு சக நிரலாளர் மறுஆய்வு செய்வது வழுக்களைக் குறைத்து நிரலின் தரத்தை மேம்படுத்தும்.

- எல்லா சோதனைகளையும் செய்து முடித்து குறுவோட்டத்தின் முடிவில் பணிகளை வெளியீட்டுக்குத் தயாராக ஆக்குங்கள்.
- ஒவ்வொரு குறுவோட்டமும் முடிந்தபின் ஆய்வு செய்து தொடர்ச்சியான மேம்பாடுகள் செயல்படுத்துங்கள்.

அடுத்து, தரம் மற்றும் நம்பகத்தன்மையை உயர்த்துங்கள். எப்படி XP பொறியியல் முறைகளை மேம்படுத்தியது என்று முன்னால் பார்த்தோம்.

- தொடர் ஒருங்கிணைப்பு செய்தல் (continuous integration).
- முதலில் ஓரலகு சோதிப்புகள் (unit tests) எழுதிவிட்டு பின்னர்தான் நிரல் எழுதவே ஆரம்பித்தல். மறுசீரமைப்பு (refactoring) செய்து நிரலை தொடர்ந்து மேம்படுத்தல். இவை இரண்டையும் சேர்த்து Test Driven

Development (TDD) என்று சொல்கிறார்கள். *மேலும் விவரங்களுக்கு இதே கணியம் இதழில் TDD பற்றிய கலாராணியின் கட்டுரைகளைப் படியுங்கள்.*

இதற்கும் மேல் உற்பத்தித்திறன் மேம்படுத்த வேண்டுமென்றால் நிறுவனத்தின் மூத்த தலைவர்கள் ஈடுபாடு கொள்ள வேண்டும் என்கிறார் ஜெஃப் சதர்லேண்ட். நிறுவனத்தின் அமைப்பு ரீதியான இடையூறுகளை நீக்கினால் திசைவேகம் மேலும் அதிகரிக்கும் என்கிறார் அவர். எடுத்துக்காட்டாக நான் முன்னால் வேலை பார்த்த ஒரு நிறுவனத்தில் செயலாக்கத் துறைக்கு ஒரு துணைத் தலைவர் மென்பொருள் வளராக்கத் துறைக்கு வேறொரு துணைத் தலைவர். ஆகையினால் ஒருங்கிணைப்பு கடினமாக இருந்தது. அனைத்துக் கோரிக்கைகளும் முறைப்படியான வழிகள் மூலமாகப் போக வேண்டியிருந்தது. வேலை செய்து வாங்குவதே கடினமானால் முன்னுரிமை பெறுவது எப்படி? அதற்குப் பதிலாக ஒரு

அமைப்பு நிர்வாகி மொய்திரள் அணியின் ஒரு பகுதியாக செயல்பட்டால் வேலையை மிகவும் எளிதாக செய்து முடிக்க இயலும்.

உங்கள் தகவெளிமை (Agile) / மொய்திரள் (Scrum) பயணம் வெற்றியடையட்டும்!

12. நேருக்கு நேர் உரையாடல்தான் சிறந்தது என்கிறார்கள், ஆனால் நாம் இருப்பதோ கடல்கடந்து!

"ஒரு மென்பொருள் உருவாக்கும் அணி கருத்துப் பரிமாற மிகவும் திறமையான மற்றும் பயனுள்ள முறை நேருக்கு நேர் உரையாடல்தான்."தகவெளிமை கொள்கை விளக்க அறிக்கையுடன் (Agile Manifesto) வெளியிடப்பட்ட மென்பொருளுக்கான கோட்பாடுகளில் இது ஒன்று.

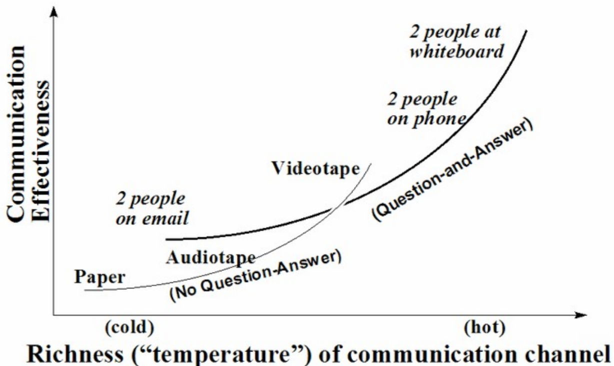
ஆனால் எப்பொழுதும் குழு உறுப்பினர்கள் யாவரும் ஒரே இடத்தில் இருப்பது இல்லை. என்னுடைய அணிகளில் இரண்டில் யாவரும் ஒரே அலுவலகத்தில் வேலை செய்தாலும்

போக்குவரத்து நெரிசல் காரணமாக வாரத்தில் பல நாட்கள் தொலைவேலை செய்ய அனுமதி உண்டு. மேலும் பெரிய நிறுவனங்களில் ஒரு குழு உறுப்பினர்கள் வெவ்வேறு அலுவலகங்களில் இருக்கலாம். அரிய திறன் உடையவர்களை வேறு ஊரில் இருந்தாலும் பணியமர்த்த நேரிடலாம். சில திட்டங்களில் வெளி நிறுவனங்களிடம் துணை ஒப்பந்தம் செய்வதும் *(outsourcing)* உண்டு. இவ்வாறு இருக்கும்போது தகவல்தொடர்பை திறம்பட நிர்வகிப்பது மொய்திரள் *(scrum)* அணிகளின் வெற்றிக்கு மிக முக்கியம்.

சுருக்கமாகச் சொல்லப்போனால் முழு அணியும் ஒரே இடத்தில் இருப்பதுதான் மென்பொருள் உருவாக்க மிக உகந்தது. ஆனால் பல்வேறு நடைமுறை காரணங்களுக்காக நாம் பரம்பிய அணிகளை பயன்படுத்த வேண்டிய கட்டாய நிலை வருகிறது. இந்த நிலைமையில் நாம் தொழில்நுட்பத்தை பயன்படுத்தி பரம்பிய அணிகளின் தகவல்தொடர்பு குறைபாடுகளை

இயன்றவரை குறைக்க முயற்சி செய்ய வேண்டும்.

இதற்கு நாம் ஆலிஸ்டேர் காக்பர்ன் (Alistair Cockburn) தயாரித்த வரைபடத்தை குறிப்பாக பயன்படுத்தலாம். இந்த வரைபடம் தகவல் பரிமாற்ற தடத்தின் செழுமையை ("வெப்பநிலை") கிடைமட்ட அச்சிலும் தகவல்தொடர்பு திறனை செங்குத்து அச்சிலும் காட்டுகிறது. உணர்ச்சிகளையும் தகவல் செழுமையையும் நன்றாகத் தெரிவித்தலை அதிக வெப்பம் என்று கூறுகிறார். இதில் இரண்டு வளைகோடுகள் உள்ளன. மேலே உள்ள வளைகோட்டில் உரையாடல் உள்ளது. கீழே உள்ள வளைகோட்டில் உரையாடல் கிடையாது, இது ஒரு வழி ஊடகம்.



இது முன்னாளில் தயாரித்த வரைபடம். இத்துடன் காணொளிக் கலந்துரையாடலை தொலைபேசிக் கலந்துரையாடலின் மேல்பக்கத்திலும் உரை அரட்டையை மின் அஞ்சலின் மேல்பக்கத்திலும் சேர்த்துக் கொள்ளுங்கள்.

இரண்டு பேர் ஒரு வெள்ளைப் பலகையில் வரைந்து நேருக்கு நேர் உரையாடுவதுதான்

அதிக வெப்பமான தடம், அதிக தகவல்தொடர்பு திறனை அளிக்கும். கொடுத்த சூழலில் நாம் குழுவினரின் தகவல்தொடர்பை வளைகோட்டின் மேல்நோக்கி கூடியவரை இதற்கு அருகில் நகர்த்த முயற்சிக்க வேண்டும். எடுத்துக்காட்டாக யாவரும் ஒரே இடத்தில் இல்லையென்றால் காணொளிக் கலந்துரையாடல் அடுத்த சிறந்த வழி. தொலைவேலை செய்பவர்களை வேலை நேரத்தில் உரை அரட்டையில் புகுபதிகை இட்டுக்கொண்டு வேலை செய்யச் சொல்லலாம். மற்ற குழு உறுப்பினர்களுக்கு எதுவும் தேவையானால் உடன் தொடர்பு கொள்ள வசதியாக இருக்கும். வெளியூரிலோ அல்லது வேறு அலுவலகங்களிலோ வேலை செய்பவர்களை முக்கிய நிகழ்ச்சிகளுக்கு, எ.கா குறுவோட்டம் திட்டமிடல், நேரடியாக வரச் சொல்லலாம்.

இது தவிர மென்பொருள் திட்டங்களில் கடல்கடந்த நாடுகளில் பணியை ஒப்படைப்பது (offshoring) நடைமுறையில் உள்ளது. தமிழ்நாட்டில் பல மென்பொருள் நிறுவனங்கள் வெளிநாடுகளுக்கு மென்பொருள் செயலி அல்லது தயாரிப்பு வேலைகள் செய்கிறார்கள். இத்திட்டங்களில் பெரும்பாலும் தயாரிப்பு உரிமையாளர், மொய்திரள் நடத்துனர் மற்றும் பங்குதாரர்கள் வெளிநாட்டிலும் ஏனைய உருவாக்குனர் குழு உறுப்பினர்கள் தமிழ்நாட்டிலும் இருப்பது சகஜம். இத்திட்டங்களில் தொலைத்தொடர்பு பிரச்சினையுடன் கலாச்சார வேறுபாடுகளும் நேர வித்தியாசமும் சேர்ந்து குழுப்பணிக்கு பெரிய சவாலாக அமைகின்றன. இந்த அணிகளில் தகவல்தொடர்பை மேம்பாடு செய்ய மேலே கூறியவை தவிர கீழ்கண்ட வழிமுறைகளையும் செயல்படுத்துவது அவசியம்.

- இரண்டு குழுக்களுக்கும் வேலை நேரம் குறைந்தபட்சம் 2 மணி நேரமாவது மேல்படிய வேண்டும்.

- அணிகளுக்கு பொதுவான நாட்காட்டி அவசியம் தேவை. அலுவலக விடுமுறை நாட்களும், குழு உறுப்பினர்கள் விடுமுறை நாட்களும், மொய்திரள் நிகழ்ச்சிகளும் யாவருக்கும் ஒரே இடத்தில் தெரிய வேண்டும். மேலும் யாவரும் ஒரே இடத்தில் ஆவணங்களை பகிரவும் திருத்தவும் இயல வேண்டும்.

- குழுவுக்கு வெளியில் தொடர்புக்கு மின் அஞ்சல் தேவைதான். ஆனால் குழுவுக்குள் தொடர்பு கொள்ள உரை அரட்டை ஒன்றைத் தேர்ந்தெடுங்கள். குழு முழுவதுமோ அல்லது தனியாகவோ தொடர்பு கொள்ள வசதி தேவை.

- கூடுதல் செலவு என்றாலும் ஒரு அணியில் இருந்து தூதுவர்களை மற்ற அணியைச் சந்திக்க அனுப்புங்கள். கலாச்சார வேறுபாடுகளைத் தாண்டி புரிதலை உருவாக்க இதுதான் முக்கிய வழி.

- இதில் மொய்திரள் நடத்துனருக்கு முக்கிய பங்கு உண்டு. கூடியவரை மொய்திரள் நடத்துனர் இரண்டு அணிகளின் மொழிகளையும் பேச இயன்றால் இரு கலாச்சாரங்களையும் இணைக்க மேலும் துணை புரியும்.

- முன்னெல்லாம் செயல்பாட்டை அடிப்படையாகக் கொண்டு வேலையைப் பிரிப்பது வழக்கமாக இருந்தது. எ.கா ஆய்வு மற்றும் வடிவமைப்பு செய்து முடித்து நிரல் எழுதும் பணியை மட்டும் கடல்கடந்த நாடுகளில் ஒப்படைப்பது. அருவி செயல்முறையில் இவ்வாறு செய்தனர். ஆனால் மொய்திரள் செயல்முறைக்கு இது

ஒத்து வராது. உருவாக்குனர் குழு அனைத்து தொழில்நுட்ப திறன்களும் கொண்டு தன்னிறைவு பெற்றிருக்க வேண்டும். அவர்களுக்கு எழும் கேள்விகளுக்கு பதில் கிடைக்க ஒரு இரவு காத்திருக்கத் தேவையில்லாமல் உடனே பதில்களைப் பெற வழி இருக்க வேண்டும்.

•தகவெளிமை செயல்முறைகள் ஆவணங்களைக் குறைத்து நிரல் எழுதி வெளியிடுவதை வலியுறுத்துகிறது என்று முன்னர் பார்த்தோம். இதற்கு எதிர்மாறாக கடல்கடந்த நாடுகளில் வேலையை ஒப்படைத்தால் ஓரளவு ஆவணங்கள் எழுதவேண்டி வருகிறது. ஏனெனில் நேருக்கு நேர் தொடர்பு குறைகிறது என்பதால் சில ஆவணங்கள் அவசியமாகின்றன.

13. தன்னமைவு மற்றும் பன்முக செயல்பாட்டுக் குழுக்களை ஊக்குவித்துத் தகவல் யுகத்துக்கு வந்து சேருங்கள்!

அப்பொழுது நான் ஒரு வணிக ஊடக நிறுவனத்தில் வேலை செய்து கொண்டிருந்தேன். நான்கு அணிகள் இருந்தன. ஒவ்வொரு அணியிலும் வடிவமைப்பாளர்கள், நிரலாளர்கள் மற்றும் சோதனையாளர்கள் இருந்தனர். இரண்டு வாரக் குறுவோட்டம், மொத்தம் பத்து வேலை நாட்கள். முதல் இரண்டு மூன்று நாட்களுக்கு வடிவமைப்பாளர்கள் மிகவும் ஓய்வில்லாமல் வேலை செய்வர். நிரலாளர்களும் சோதனையாளர்களும் வடிவமைப்புகள் தயாராகி பங்குதாரர்கள்

ஒப்புதல் தரும் வரை ஏதாவது செய்து கொண்டிருப்பார்கள். ஆனால் உற்பத்தித்திறன் குறைவு. எவ்வாறு இருக்க முடியும்? அடுத்த இரண்டு மூன்று நாட்களுக்கு நிரலாளர்களுக்கு ஓய்வற்ற வேலை. கடைசி இரண்டு மூன்று நாட்களுக்கு சோதனையாளர்களுக்கு ஓய்வற்ற வேலை. அவர்கள் ஏதாவது வழு கண்டுபிடித்தால் பெரும்பாலும் நிரலாளர்கள் சரி செய்ய வேண்டி வரும். சில சமயங்களில் வடிவமைப்பாளர்களும் சரி செய்ய வேண்டி வரலாம்.

ஆக நாங்கள் Scrum-ன் குறுவோட்டத்துக்குள் ஒரு சிறிய அருவி செயல்முறை நடத்திக் கொண்டிருந்தோம்! இவ்வாறு ஏன் நடக்கிறது என்று புரிந்து கொள்ள நாம் சரித்திரத்தை கொஞ்சம் புரட்டிப் பார்க்கவேண்டும்.

தொழிற்புரட்சிக்கு முன்னர் பொருள் உற்பத்திப் பணிகள் திறமையான கைவினைஞர்களால் செய்யப்பட்டன என்பதை நாம் அறிவோம். எடுத்துக்காட்டாக சோழர் காலத்தில் ஸ்தபதிகள்,

சிற்பிகள், கொல்லர்கள், தச்சர்கள், கற்தச்சர்கள், பொற்கொல்லர்கள் என்று பல கைவினைஞர்கள் இருந்தனர். தொழிற்புரட்சியின்போதுதான் உழைப்புப் பிரிவினை *(division of labour)* வந்தது. முன்பெல்லாம் ஒரு முழுப் பொருளையும் உருவாக்கிய ஒரு தொழிலாளி இப்போது அந்தப் பொருளின் ஒருபகுதியை மட்டுமே உற்பத்தி செய்வதால் கைவினை பயிற்சிக்குத் தேவையான நீண்ட பயிற்சி காலத்தைக் குறைக்க முடிந்தது. இந்த உழைப்புப் பிரிவினை உற்பத்தித்திறனையும் உயர்த்தியது.

இந்த உழைப்புப் பிரிவினையின் விளைவுகளைத்தான் நாம் மேலே பார்க்கிறோம். நிரலாளர்களும், வடிவமைப்பாளர்களும் சோதனை செய்ய விரும்புவதில்லை. சோதனையாளர்களுக்கு நிரல் எழுதவும், வடிவமைக்கவும் தெரியாது. மென்பொருளை உருவாக்கி வெளியீடு செய்ய இந்த மூன்று பங்களிப்புகளிலும் நேரடியாக வராத பல பணிகளும் இருக்கலாம். யார் இந்த பணிகளைச்

செய்வது? மேலும் முக்கியமாக யார் இந்த பணிகளை இன்னின்னார் செய்ய வேண்டும் என்று முடிவு செய்து ஒதுக்குவது? மேற்பார்வையாளர்களுக்கும் மேலாளர்களுக்கும் இதுதான் முக்கிய வேலையாக உள்ளது. ஒதுக்கப்படும் வேலையை வேண்டா வெறுப்பாக செய்கிறார்கள். அணியின் மன உறுதியோ கீழே செல்கிறது.

தன்னார்வலர்களும் துளிர் நிறுவனங்களின் (Startups) அணிகளும் இவ்வாறு இது தங்கள் வேலை இது தங்கள் வேலையல்ல என்று பிரித்துப் பார்ப்பதில்லையே. யாரால் எந்த வேலையைச் செய்ய முடியுமோ அதைத் தாங்களாகவே எடுத்துக்கொண்டு செய்து முடிக்கிறார்கள் அல்லவா?

பறவைகள் கூட்டம்: உயிரியலில் தன்னமைவுக்கு ஒரு எடுத்துக்காட்டு.

இக்காரணத்தினால்தான் மொய்திரள் குழுக்கள் பன்முக செயல்பாடு மற்றும் தன்னமைவுக் குழுக்கள் என்று மொய்திரள் கையேடு சொல்கிறது. தன்னமைவுக் குழுக்கள் என்றால் என்ன? அணியின் வெளியே இருப்பவர்கள் இயக்குவதற்குப் பதிலாக, தன்னமைவுக் குழுக்கள் தங்கள் வேலையை எப்படி சிறப்பாக நிறைவேற்றுவதென்று தாங்களே முடிவு செய்வதுதான்.

“பழைய” கோட்பாடுகள் பொறியாளர்கள் மற்றும் பொருளாதார அறிஞர்களால் உருவாக்கப்பட்டவை. நிறுவனங்கள் நல்ல எண்ணெய் போட்ட இயந்திரம் போல சீராக இயங்க வேண்டும் என்று அவர்கள் நம்பினர். தொழிற்துறை யுகத்தில் இருந்த நடவடிக்கைகளுக்கு இந்த மேலாண்மை மாதிரிகள் குறைந்தபட்சம் போதுமானவையாக இருந்தன.



அந்தக் காலகட்டத்தில் சுற்றுச்சூழல் ஓரளவு நிலையாக இருந்தது. திரும்பத்திரும்ப ஒரே மாதிரியான வேலை உடல் உழைப்பால் நடந்தது. புதிய கருத்துக்கள் மற்றும் உத்திகளை உருவாக்குவது அனேகமாக முழுவதும் உயர்மட்ட நிர்வாகத்தின் கையில் இருந்தது.

மனிதர்கள் இயந்திரங்கள் அல்லவே. மேலும் நாம் தகவல் யுகத்துக்கு வந்தபின்

நிறுவனங்களின் பொருளாதாரம் பெரும்பாலும் அறிவுசார் மூலதனங்களை நம்பியே உள்ளன. இந்த அறிவின் மதிப்பு தினசரி அதிகரிக்கும் நிலையில் நாம் அறிவு ஊழியர்களை பாரம்பரிய முறையில் நிர்வகிக்கக் கூடாது என்பதைப் புரிந்து கொள்ள ஆரம்பித்திருக்கிறோம். அறிவு ஊழியர்களை சொன்னபடி செய்ய வேண்டுமென்று நீங்கள் கட்டாயப்படுத்தினால் அவர்களது படைப்பாற்றலை இழந்து விடுவீர்கள்.

எனினும், இம்மாதிரி ஒரு இறுக்கமான கட்டளை மற்றும் கட்டுப்பாட்டு முறையில் இயங்கி வந்த நிறுவனங்களில் தன்னமைவுக் குழுக்களாக செயல்படுவது என்பது சொல்வது எளிது, ஆனால் நடைமுறையில் செயல்படுத்துவது கடினமே. அணி மேலாளர்கள் எப்படி வேலை செய்யப்படுகிறது என்பதில் தங்களுக்கு இருக்கும் இறுக்கமான பிடியை விட மிகத் தயங்குவார்கள். மேலும் குழு

உறுப்பினர்கள் மேலிடத்தில் இருந்து வழிகாட்டலை எதிர்பார்த்து காத்திருப்பார்கள்.

அப்படியென்றால் தானாகவே வேலை நடக்குமா? தலைமையே தேவை இல்லையா என்று கேட்கிறீர்களா? அப்படியே விட்டால் யார் எதற்கு பொறுப்பு என்பதே தெரியாமல் குழப்பம் நிலவும். தலைக்குத் தலை நாட்டாமையாக ஆகி விடலாம்.

மாறாக, திறமையான மேலாண்மை மிக அவசியம் தேவை. கட்டளை கட்டுப்பாடு பாணி மேலாண்மையைத் தவிர்க்க வேண்டும். ஆனால் சுயேச்சையான படைப்பாற்றல் கொண்ட ஊழியர்கள் இடையே உருவாகும் நடத்தை முறைகளின் பரிணாம வளர்ச்சியை சரியான வழியில் ஆற்றுப் படுத்துவது முக்கியம்.

தங்களது குழுவில் யார் யார் வேலை செய்வது என்று தேர்ந்தெடுப்பதையும் குழுவே செய்ய வேண்டும் என கென் ஷ்வாபர் கூறுகிறார். ஆனால் இது தகவெளிமை (Agile) / மொய்திரள்

(Scrum) நடைமுறையைப் பயிற்சி செய்து முதிர்ந்த அணிகளுக்கான அறிவுரை. புதிதாக இவ்வழியில் அடியெடுத்து வைப்பவர்கள் முயற்சி செய்ய வேண்டாம்.

நேரடியான கட்டளை கட்டுப்பாட்டில் இறங்காமல் ஆனால் அதே நேரத்தில் செல்வாக்குடன் இருக்கும் சாமர்த்தியம் தகவெளிமை குழுக்களின் தலைவர்களுக்குத் தேவை. ஒரு ஆதரவான சூழ்நிலையை உருவாக்குவதாலும், தேவையான எல்லைகள் மற்றும் கட்டுப்பாடுகள் அமைப்பதாலும், நிறுவன இடையூறுகளை நீக்குவதாலும், நிறுவனத்தின் மற்ற துறைகளுடன் இணைக்கும் பாலமாக செயல்படுவதாலும் இந்த செல்வாக்கை அவர்கள் அடையலாம்.

14. பயனர் கதையை தெளிவாகத் தயார் செய்தால் பாதி வேலையை முடித்தது போல!

நாம் முன்னர் பார்த்தபடி, மென்பொருள் தேவைகள் பட்டியல் ஒரு தகவல் தொடர்பு பிரச்சினை. மென்பொருள் உருவாக்கி வாங்க விரும்புபவர்கள் அதை உருவாக்கத் தெரிந்தவர்களுக்கு தெளிவாகச் சொல்வது அவசியம். இல்லாவிட்டால் பிள்ளையார் பிடிக்கப் போய் குரங்காய் முடிந்தது என்று சொல்கிறார்களே அம்மாதிரி ஆகிவிடும்.

70 – 80 களில் அமெரிக்க வங்கிகளுக்கான கட்டுப்பாடுகள் தளர்த்தப்பட்டன. அவர்கள் மற்ற வங்கிகளுடன் போட்டியில் வெல்ல என்ன

செய்யலாம் என்று சிந்திக்கத் தொடங்கினர். இதன் பொருட்டு தங்கள் வாடிக்கையாளர்களிடம் கருத்துக் கணிப்பு செய்யத் தொடங்கினர். இந்த கருத்துக் கணிப்பில் வாடிக்கையாளர்களிடமிருந்து திரும்பத் திரும்ப வந்த பின்னூட்டங்கள் வங்கிகள் அதிக மணி நேரம் திறந்திருக்க வேண்டும் மற்றும் அதிக பணப் பொறுப்பாளர்களை வேலைக்கு அமர்த்த வேண்டும் என்பவையே.

வங்கிகள் இந்த பரிந்துரைகளை அப்படியே செயல்படுத்தியிருந்தால் செலவு எக்கசக்கமாக ஆகியிருக்கும். அவர்கள் அதிகம் செலவு செய்யாமல் எவ்வாறு தீர்வு காண முடியும் என்று கேட்டனர். இதன் விளைவாகத்தான் 60 களிலேயே கண்டுபிடிக்கப்பட்டு ஆனால் சந்தையில் பயன்படுத்தப்படாமல் கிடந்த ஏடிஎம் (ATM) கருவி பரவலாக நிறுவப்பட்டது.

மேற்கண்ட கதையின் உட்கருத்து என்னவென்றால் தேவைகள் பட்டியல் பயனர்

கண்ணோட்டத்தில் இருந்து எழுதப்பட வேண்டும். மற்றும் பயனர்களுக்கு "என்ன" தேவை என்பது மட்டுமல்லாமல் "ஏன்" அது தேவை என்பதையும் தெளிவாக்குவது அவசியம். ஆனால் "எப்படி" செயல்படுத்துவது என்பதுபற்றி எதுவும் இருக்கக் கூடாது. புதுமையான வழிகளில் பயனர் இலக்குகளை அடைவதற்கு உருவாக்குநர்களுக்கு இணங்கு தன்மை வேண்டும்.

நிகழ்பட விளையாட்டு வீரராகிய நான் சிறுகோள்களில் மோதாமல் தவிர்க்க இடது மற்றும் வலது அம்புக்குறியை அழுத்தும்போது என் விண்கலம் முன்னும் பின்னுமாக நகர வேண்டும்.

ஏற்றுக் கொள்வதற்கான நிபந்தனைகள்: ...

அருவி செயல்பாட்டில் நன்கு எழுதப்பட்ட தேவைகள் பட்டியல்கள் "எப்படி" என்பதைத் தவிர்த்து "என்ன" என்பதில் கவனம் வைப்பது உண்டு. ஆனால் "ஏன்" அது தேவை என்பதை ஒவ்வொரு தேவையிலும் விளக்குவது அரிதே. தகவெளிமை *(Agile)* / மொய்திரள் *(Scrum)* செயல்முறைகளில் தேவைகள் பட்டியலை பயனர் கதைகள் என்று கூறுகிறோம். பயனர் கதைகளை "<இன்ன வகை> பயனராகிய நான் <இன்ன காரணங்களால்> <இன்ன இலக்கை> அடைய வேண்டும்." என்ற வடிவில் எழுதுவது பரிந்துரைக்கப்படுகிறது.

ஒரு குறுவோட்டம் முடியும் தருவாயில் மொய்திரள் குழுவினர் அடுத்த குறுவோட்டத்துக்கு கிடக்கும் பணிகளைத் தெளிவாக்குவதற்காக கூடிப்பேச வேண்டும். இக்கூட்டத்தில் குழு மற்றும் தயாரிப்பு உரிமையாளர் கிடக்கும் பணிகள் பட்டியலில் மேல் வரிசையில் உள்ள பணிகளைப்பற்றி கலந்தாய்வு செய்ய வேண்டும். திட்டமிடும்

போது எழும் கேள்விகளை முன்னரே கேட்க குழுவினருக்கு இது ஒரு நல்ல வாய்ப்பு. இந்த கேள்விகளை முன்னரே கேட்பதன் மூலம் உடனடியாக பதில் இல்லை என்றால் தயாரிப்பு உரிமையாளர் திட்டமிடுவதற்கு முன்னால் அவற்றை சேகரிக்க இயலும்.

ஒரு பயனர் கதைக்கு பல லட்சணங்கள் உள்ளன. அவற்றில் முக்கியமான சில: ஒவ்வொரு கதையும் மென்பொருளின் பயனருக்கு மதிப்புள்ள பயனை வழங்க வேண்டும். அதிகபட்சம் ஒரு குறுவோட்டத்தில் முடிக்கும் அளவுக்கு சிறிதாக இருக்க வேண்டும். முடிந்தால் அதை விட சிறியதாக இருப்பதே மேல். பல கதைகளை உள்ளடக்கிய பெரிய தேவைகளை காப்பியம் என்று கூறுகிறோம். அத்தகைய காப்பியங்களை உடைத்து, சமாளிக்கக்கூடிய அளவில் கதைகளாக எழுத வேண்டும். பயனர் கதை அல்லது அதன் தொடர்புடைய விளக்கம் அதை சோதனை செய்யத் தேவையான தகவல்களை வழங்க

வேண்டும். இவற்றை ஏற்றுக் கொள்வதற்கான நிபந்தனைகள் என்று தனியாக பயனர் கதையில் எழுதுவது வழக்கம். ஏற்றுக் கொள்வதற்கான நிபந்தனைகள் அளவிட சாத்தியமாக இருப்பது நல்லது. எதிர்பார்த்தபடி செயல்படுகிறதா என்று தீர்மானிக்க ஒரு சாத்தியமான வழி இருந்தால் அதை சோதனை செய்யக்கூடியது என்று கூறலாம்.

அருவி செயல்முறையில் தேவைகள் பட்டியல் பங்குதாரரர்கள் மற்றும் உருவாக்குநர் குழு இடையே உள்ள ஒப்பந்தம் ஆகும். மாறாக மொய்திரளில் இவ்வாறு எழுதிய கதைகள் தயாரிப்பு உரிமையாளர் மற்றும் உருவாக்குநர் குழு இடையே ஒரு விரிவான உரையாடலை ஆரம்பிக்கும் வழி என்றே கருதப்படுகின்றன. கூடுதல் விளக்கங்கள் தயாரிப்பு உரிமையாளரால் உரையாடல்களில் வழங்கப்படும்.

கிடக்கும் பணித்தொகுதிகளில் அம்சங்கள் மற்றும் கதைகளை சேர்ப்பதற்கு தயாரிப்பு உரிமையாளர்தான் பொறுப்பு. ஆனால்

உருவாக்குநர் குழு தயாரிப்பு உரிமையாளருடன் கூடி வேலை செய்ய வேண்டும். குறைந்தபட்சம் அடுத்த குறுவோட்டத்துக்கான கதைகளை உடனடியாக நடவடிக்கை எடுக்கக்கூடிய வடிவத்தில் கொண்டு வர வேண்டும். மேலும் தயாரிப்பு உரிமையாளர், குறுவோட்டம் தொடங்கும் வரை பயனர் கதைகளில் தேவைப்பட்ட மாற்றங்களை செய்யலாம். எடுத்துக்காட்டாக உருவாக்குநர் குழு ஒரு கதைக்கு அதிக முயற்சி எடுக்குமென்று மதிப்பீடு செய்தால் தயாரிப்பு உரிமையாளர் அந்தக்கதையில் சில அம்சங்களை குறைக்கலாம் அல்லது அந்தக்கதையையே கிடக்கும் பணிகள் பட்டியலில் கீழே நகர்த்தி விடலாம்.

பாதுகாப்பு போன்ற பொதுவான தேவைகளை செயல்பாடு அல்லாத தேவைகள் *(non-functional requirements)* என்று கூறுகிறோம். இவற்றை ஒவ்வொரு பயனர் கதையிலும் எழுதத்தேவையில்லை. இவற்றை பொதுவாக ஏற்றுக்கொள்ளப்பட்ட வரையறையில் *(definition*

of done) ஒரு பகுதியாக சேர்த்துவிடுவோம். பொதுவாக ஏற்றுக்கொள்ளப்பட்ட வரையறை பற்றி நாம் பின்னர் வரும் கட்டுரையில் விரிவாகப் பார்ப்போம்.

15. மொய்திரளில் வேலையின் அளவை மதிப்பீடு செய்வது இன்னொரு வகையான சூதாட்டமா?

மொய்திரள் (Scrum) செயல்முறையின் சக படைப்பாளரான ஜெஃப் சதர்லேண்ட் (Jeff Sutherland) கூறுகிறார், "நான் OpenView Venture Partners கூட வேலை செய்த பொழுது அவர்கள் எந்த ஒரு இயக்குநர் குழுமம் கூட்டத்திலும் சரியான கான்ட் விளக்கப்படம் பார்த்ததில்லை என்று கூறுவர். தங்கள் அணிகளின் உற்பத்தி திசைவேகம் என்ன என்றே தெரியாமல் இன்ன தேதியில் வெளியீடு செய்ய முடியும் என்று வாக்குறுதி அளிப்பதுதான் இத்திட்டங்களின்

தோல்விக்கு மூல காரணம். மொய்திரள் செயல்படுத்துவதற்கு முன் இருந்த நிலை இது.”
நாம் அணியின் திசைவேகம் மதிப்பிட இரண்டு முக்கியமான காரணங்கள் உள்ளன:

- ஒன்று – நாம் முன்னர் பார்த்தபடி வெளியீட்டுத் தேதிக்கு தயாரிப்பு உரிமையாளர்தான் பொறுப்பு. கிடக்கும் பணிகளின் மதிப்பீடும், அணியின் திசைவேகமும் தெரிந்தால்தான் தயாரிப்பு உரிமையாளர் சாத்தியமான வெளியீட்டுத் தேதி முடிவு செய்ய இயலும்.

- இரண்டு – நாம் மென்பொருள் உருவாக்கும் வழிமுறை மேம்பாடு செய்தபின் திசைவேகம் அதிகரித்ததா இல்லை குறைந்ததா என்பதை அளவிட இயல வேண்டும். அணியின் திசைவேகம் தெரியாது என்றால் மேம்பாடுகள் வேலை செய்ததா என்பதைத் தெரிந்து கொள்வது மிகக் கடினம்.

எப்படி மொய்திரள் இம்மாதிரி பிரச்சினைகளைத் தாண்டி சாத்தியமான வெளியீட்டுத் திட்டங்களைத் தயாரிக்க உதவுகிறது? வேலை உள்ளடக்கத்தை அளவிட பயன்படுத்தப்படும் அலகு எது? அது எப்படி பிற முறைகளில் உள்ள குறைபாடுகளை நிவர்த்தி செய்கிறது? நாம் இத்தகைய கேள்விகளுக்கு பதிலளிக்க இங்கு முயற்சி செய்வோம்.

ஒரு பயனர் கதையை செய்து முடிக்க எத்தனை மணி நேரம் எடுக்கும் என்று மதிப்பீடு செய்வதில் கீழ்க்கண்ட பிரச்சினைகள் உள்ளன:

- ஆராய்ச்சிகளின்படி மணி நேரத்தை மதிப்பீடு செய்வது நமக்கு இயல்பாகவே கடினம் என்று தெரிகிறது. நாம் ஒரு வேலைக்கு எவ்வளவு நேரம் எடுக்கும் என்று மதிப்பிடும்போது 400% வரை பிழை வரலாம். நேரடி மதிப்பீட்டை விட ஒப்பிடும் மதிப்பீடே எளிதானது. ஒன்று மற்றொன்றை விட

பெரியது அல்லது சிறியது என்று ஓரளவு துல்லியமாக நம்மால் சொல்ல இயலும்.

- ஒரு குழுவில் எல்லோருக்கும் நிரலாக்கத்தில் அதே அளவு திறமை இருக்காது. ஆகவே ஒருவேலைக்கு எத்தனை மணி நேரம் எடுக்கும் என்பதில் முடிவற்ற வாக்குவாதங்கள் எழக்கூடும்.

- மணி நேரம் போன்ற அளவீடுகளைத் தவறாகப் புரிந்துகொள்ள வாய்ப்புகள் அதிகம். எடுத்துக்காட்டாக, செய்த மணி நேரத்தை மதிப்பீட்டு மணி நேரத்துடன் ஒப்பிட்டு சில மேலாளர்கள் பகுப்பாய்வு செய்கிறார்கள். இதன் மூலம் இன்னார் நன்றாக வேலை செய்கிறார்கள், இன்னார் நன்றாக வேலை செய்யவில்லை என்ற முடிவுக்கு வருகிறார்கள். இது சரியல்ல. ஏனெனில் நாம் முன்னர் பார்த்தபடி மென்பொருள் வேலையில் தேவைகள்

மாறக்கூடியவை, மற்றும் தொழில்நுட்பம் நிச்சயமற்றது.

மேற்கண்ட குறைபாடுகளை நிவர்த்தி செய்வதற்காக ஜெஃப் சதர்லேண்ட் *(Jeff Sutherland)* கதைப் புள்ளிகள் *(Story points)* என்ற புதிய அலகை அறிமுகம் செய்துவைத்தார். கதைப் புள்ளிகள் பற்றிய முக்கிய அம்சங்கள்;

- இவை ஒப்புநோக்கு மதிப்பீடுகள். முன்னால் செய்த கதைகளுடன் ஒப்பிட்டு புதிய கதைகளை மதிப்பீடு செய்ய வேண்டும்.

- ஒரு அணியின் மதிப்பீட்டை மற்றொரு அணியின் மதிப்பீட்டுடன் ஒப்பிட இயலாது.

- ஒரு கதைப் புள்ளிக்கு ஒரு அணி சராசரியாக எவ்வளவு நேரம் எடுக்கும் என்று நிச்சயமாகக் கணக்கிட முடியும். எனினும் ஒரு குறிப்பிட்ட கதையை முடிக்க இவ்வளவு நேரம்தான் எடுக்கும் என்று எவரும் சொல்ல இயலாது.

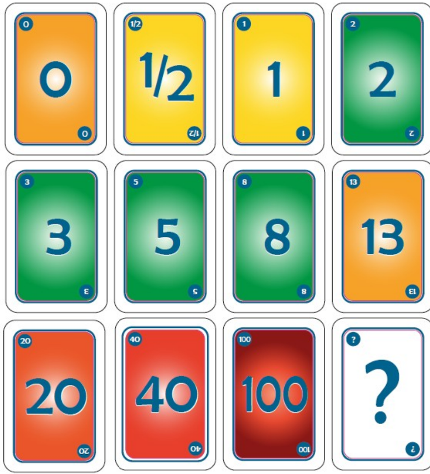
- இரண்டு புள்ளிகள் என்று நாம் மதிப்பீடு செய்த ஒரு கதைக்கு ஒரு புள்ளி என்று மதிப்பீடு செய்த இன்னொரு கதை அளவே நேரம் எடுக்கலாம். அல்லது அதற்கு மூன்று புள்ளி என்று மதிப்பீடு செய்த மற்றொரு கதை அளவு நேரமும் எடுக்கலாம். எனினும், எல்லாவற்றையும் ஒன்று சேர்த்துப் பார்க்கும்போது இந்த தனிப்பட்ட ஏற்ற இறக்கங்கள் ஒன்றுக்கொன்று ரத்து ஆகி விடும். எனவே நீண்ட கால அடிப்படையில் கதைப் புள்ளிகளை அணியின் திசைவேகம் அளவிட பயன்படுத்தலாம்

எனினும் கதைப் புள்ளிகளில் ஒரு குறைபாடு இருக்கத்தான் செய்கிறது. நாம் இந்தத் தொடரில் முன்னர் பார்த்தபடி சதுர அடியில் கட்டிடத்தின் பரப்பளவு போன்ற வெளியீட்டை அளவிடும் அலகே சிறந்தது. ஆனால் கதைப் புள்ளிகள் எனும் இந்த அலகு எவ்வளவு வேலை உள்ளீடு செய்கிறோம் எனபதற்கான அலகு.

எப்பொழுது இந்த மதிப்பீடு செய்வது? ஒரு குறுவோட்டம் முடியும் தருவாயில் கிடக்கும் பணிகளைத் தெளிவாக்குவதற்காக கூடிப்பேச வேண்டும் என்று நாம் முன்னர் பார்த்தோம். இக்கூட்டத்தில் மதிப்பீடு செய்வது ஒரு முக்கிய வேலை.

யார் மதிப்பீடு செய்ய வேண்டும்? இந்த வேலையை யார் செய்யப் போகிறார்களோ அவர்கள்தான் மதிப்பீடு செய்யவேண்டும். அதாவது உருவாக்குனர்கள் குழு. அனைவரும் பங்கு பெற வேண்டிய அவசியம் இல்லை. சிலருக்கு முடிக்க வேண்டிய வேலை இருந்தால் மற்றவர்கள் பங்கு பெறலாம்.

எப்படி செய்ய வேண்டும்? இதற்கு



பரிந்துரைக்கப்படும் முக்கியமான செயல்முறை மதிப்பீடு சீட்டாட்டம் (Planning Poker). இந்த செயல்முறையின் முக்கிய அம்சம் அணி முன்னர் செய்த வேலைகளுடன் ஒப்பீடு செய்து ஒருமித்த கருத்தால் மதிப்பிட வேண்டும்.

குறிப்பு: நாம் "பயனர் கதை' (user story) அல்லது 'கதை' என்று சொல்வதை தயாரிப்புக்கான கிடக்கும் பணிகளில் ஒன்று (Product Backlog Item or PBI) என்றே மொய்திரள் கையேடு குறிப்பிடுகிறது. கதை என்பது அதீத நிரலாக்கத்தில் (Extreme Programming or XP) ஆக்கிய சொல். படிக்கவும், விவாதிக்கவும் எளிதாக இருக்கும் என்பதால் இதையே இக்கட்டுரைத் தொடரில் பயன்படுத்துகிறோம்.

உங்கள் தகவெளிமை (Agile) / மொய்திரள் (Scrum) பயணம் வெற்றியடையட்டும்!

முற்றும்

நன்றி

கணியம் பற்றி

இலக்குகள்

- கட்டற்ற கணிநுட்பத்தின் எளிய விஷயங்கள் தொடங்கி அதிநுட்பமான அம்சங்கள் வரை அறிந்திட விழையும் எவருக்கும் தேவையான தகவல்களை தொடர்ச்சியாகத் தரும் தளமாய் உருபெறுவது.

- உரை, ஒலி, ஒளி என பல்லூடக வகைகளிலும் விவரங்களை தருவது.

- இத்துறையின் நிகழ்வுகளை எடுத்துரைப்பது.

- எவரும் பங்களிக்க ஏதுவாய் யாவருக்குமான நெறியில் விவரங்களை வழங்குவது.

- அச்சு வடிவிலும், புத்தகங்களாகவும், வட்டுக்களாகவும் விவரங்களை வெளியிடுவது.

பங்களிக்க

- விருப்பமுள்ள எவரும் பங்களிக்கலாம்.
- கட்டற்ற கணிநுட்பம் சார்ந்த விஷயமாக இருத்தல் வேண்டும்.
- பங்களிக்கத் தொடங்கும் முன்னர் கணியத்திற்கு உங்களுடைய பதிப்புரிமத்தை அளிக்க எதிர்பார்க்கப்படுகிறீர்கள்.
- editor@kaniyam.com முகவரிக்கு கீழ்க்கண்ட விவரங்களடங்கிய மடலொன்றை உறுதிமொழியாய் அளித்துவிட்டு யாரும் பங்களிக்கத் தொடங்கலாம்.
    - **மடலின் பொருள்:** பதிப்புரிமம் அளிப்பு
    - மடல் உள்ளடக்கம்
        - என்னால் கணியத்திற்காக அனுப்பப்படும் படைப்புகள் அனைத்தும் கணியத்திற்காக

முதன்முதலாய் படைக்கப்பட்டதாக உறுதியளிக்கிறேன்.

- இதன்பொருட்டு எனக்கிருக்கக்கூடிய பதிப்புரிமத்தினை கணியத்திற்கு வழங்குகிறேன்.

- உங்களுடைய முழுப்பெயர், தேதி.

- தாங்கள் பங்களிக்க விரும்பும் ஒரு பகுதியில் வேறொருவர் ஏற்கனவே பங்களித்து வருகிறார் எனின் அவருடன் இணைந்து பணியாற்ற முனையவும்.

- கட்டுரைகள் மொழிபெயர்ப்புகளாகவும், விஷயமறிந்த ஒருவர் சொல்லக் கேட்டு கற்று இயற்றப்பட்டவையாகவும் இருக்கலாம்.

- படைப்புகள் தொடர்களாகவும் இருக்கலாம்.

- தொழில் நுட்பம், கொள்கை விளக்கம், பிரச்சாரம், கதை, கேலிச்சித்திரம், நையாண்டி எனப் பலசுவைகளிலும் இத்துறைக்கு

பொருந்தும்படியான ஆக்கங்களாக இருக்கலாம்.

•தங்களுக்கு இயல்பான எந்தவொரு நடையிலும் எழுதலாம்.

•தங்களது படைப்புகளை எளியதொரு உரை ஆவணமாக editor@kaniyam.com முகவரிக்குஅனுப்பிவைக்கவும்.

•தள பராமரிப்பு, ஆதரவளித்தல் உள்ளிட்ட ஏனைய விதங்களிலும் பங்களிக்கலாம்.

•ஐயங்களிருப்பின் editor@kaniyam.com மடலியற்றவும்.

விண்ணப்பங்கள்

- கணித் தொழில்நுட்பத்தை அறிய விழையும் மக்களுக்காக மேற்கொள்ளப்படும் முயற்சியாகும் இது.

- இதில பங்களிக்க தாங்கள் அதிநுட்ப ஆற்றல் வாய்ந்தவராக இருக்க வேண்டும் என்ற கட்டாயமில்லை.

- தங்களுக்கு தெரிந்த விஷயத்தை இயன்ற எளிய முறையில் எடுத்துரைக்க ஆர்வம் இருந்தால் போதும்.

- இதன் வளர்ச்சி நம் ஒவ்வொருவரின் கையிலுமே உள்ளது.

- குறைகளிலிருப்பின் முறையாக தெரியப்படுத்தி முன்னேற்றத்திற்கு வழி வகுக்கவும்.

வெளியீட்டு விவரம்

ஆசிரியர்: த. ஸ்ரீநிவாஸன் – editor@kaniyam.com
வெளியீட்டாளர்: ம. ஸ்ரீ ராமதாஸ், 1 அக்ரஹாரம், துகிலி – 609804 தொ. பே: +91 94455 54009
– amachu@kaniyam.com

கட்டுரைகளில் வெளிப்படுத்தப்படும் கருத்துக்கள் கட்டுரையாசிரியருக்கே உரியன.

நன்றி